НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет Інформатики та обчислювальної техніки
Обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій, СТІРЕНКО
«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Веб-сервіс для моніторингу Інтернет-медіа ресурсів»

Виконав:
студент IV курсу, групи ІО-64
Траєр Артем Михайлович

_____

Керівник:
доктор технічних наук, професор
Болдак Андрій Олександрович

_____

Консультант з нормоконтролю:
доктор технічних наук, професор
Сімоненко Валерій Павлович

_____

Рецензент:
кандидат технічних наук, доцент
Катерина Ліщук Миколаївна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
"__"_____20__р.

ЗАВДАННЯ

на дипломну роботу студенту

Траєра Артема Михайловича

1. Тема проекту (роботи) «Веб-сервіс для моніторингу Інтернет-медіа ресурсів», керівник проекту (роботи) Болдак Андрій Олександрович, кандидат технічних наук, доцент, затверджені наказом по університету від " 07" травня 2020 р. № 1081-с

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вихідні дані до роботи: технічна документація, теоретичні дані;

4. Зміст пояснювальної записки: основи системи моніторингу та огляд існуючих рішень, проектування системи моніторингу, розробка системи моніторингу, тестування та результати системи моніторингу;

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): схема алгоритму веб-сервісу для моніторингу інтернет-медіа ресурсів, схема структури веб-сервісу для моніторингу інтернет-медіа ресурсів, діаграма класів веб-сервісу для моніторингу інтернет-медіа ресурсів.

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	д.т.н., проф. Сімоненко В. П.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітки
1.	Затвердження теми роботи	01.09.2019	
2.	Вивчення та аналіз завдання	20.01.2020 - 09.02.2020	
3.	Розробка загальної структури системи	10.02.2020 – 01.03.202	
4.	Програмна реалізація системи	02.03.2020 – 26.04.2020	
5.	Тестування та виправлення помилок	27.04.2020 – 03.05.2020	
6.	Оформлення пояснювальної записки	04.05.2020 – 26.05.2020	
7.	Передзахист	26.05.2020	
8.	Захист	15.06.2020	

Студент Артем ТРАЄР

Керівник Андрій БОЛДАК

АНОТАЦІЯ

В дипломній роботі розроблюється он-лайн сервіс для моніторингу інтернет-медіа ресурсів, за допомогою якого за різними критеріями пошуку можно отримувати необхідну інформацію, яка буде надсилатися сервісом на почту.

Даний сервіс допомагає не витрачати багато часу на пошук необхідної інформації, а сконцентруватися лише на необхідних речах. Лише вказавши частину тексту, можна отримати від сервіса перелік посилань де дана частина зустрічається, що дозволяє зробити пошук інформації в інтернеті ефективним і швидким.

АННОТАЦИЯ

В дипломной работе разрабатывается он-лайн сервис для мониторинга интернет-медиа ресурсов, с помощью которого по различным критериям можно получать необходимую информацию, которая будет направляться сервисом на почту.

Данный сервис помогает не тратить много времени на поиск необходимой информации, а сконцентрироваться только на необходимых вещах. Только указав часть текста, можно получить сервиса перечень ссылок где данная часть встречается, что позволяет сделать поиск информации в интернете эффективным и быстрым

ANNOTATION

In this thesis, an online service for monitoring Internet media resources is being developed, with the help of which, according to various search criteria, it is possible to obtain the necessary information that will be sent by the service to the mail.

This service helps you not to spend a lot of time searching for the necessary information, but to focus only on the necessary things. Only by specifying part of the text, you can get from the service a list of links where this part occurs, which allows you to search for information on the Internet efficiently and quickly

Опис альбому
до дипломного проекту

на тему: «Веб-сервіс для моніторингу Інтернет-медіа ресурсів»

Київ – 2020

№ рядка	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість	Примітка
			Завдання на дипломний проект	2	
1	А4	ІАЛЦ.467100.001 ОА	Опис альбому	1	
2	А4	ІАЛЦ.467100.002 ТЗ	Технічне завдання	3	
3	А4	ІАЛЦ.467100.003 ПЗ	Пояснювальна записка	63	
4	А4	ІАЛЦ.467100.004 Д1	Додаток А	1	
5	А4	ІАЛЦ.467100.005 Д2	Додаток Б	1	
6	А4	ІАЛЦ.467100.006 Д3	Додаток В	1	
7	А4	ІАЛЦ.467100.007 Д4	Додаток Г	6	

Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467100.001 ОА		
Розроб.		Траєр А.М.			Веб-сервіс для моніторингу Інтернет-медіа ресурсів	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Болдак А.О.				Т	
Н. контр.		Сімоненко В.П.			Опис альбому	НТУУ «КПІ імені Ігоря Сікорського» ФІОТ Група ІО-64	
Затв.							

Технічне завдання
до дипломного проекту

на тему: «Веб-сервіс для моніторингу Інтернет-медіа ресурсів»

Київ – 2020

ЗМІСТ

1. НАЙМЕНУВАННЯ І ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування: веб-сервіс для моніторингу Інтернет-медіа ресурсів.

Область застосування: альтернатива існуючим системам та сервісам для пошуку та моніторингу інформації.

2. ПІДСТАВИ РОЗРОБКИ

Підставою для розробки є завдання на створення єдиної системи для пошуку та моніторингу інформації з різних джерел з можливістю отримання результатів на почту.

3. МЕТА І ПРИЗНАЧАННЯ РОЗРОБКИ

Метою даного проекту є створення єдиної системи для пошуку та моніторингу інформації з різних джерел з можливістю отримання результатів на почту.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є технічна література з питань розробки систем та застосування бібліотек, документації існуючих систем, публікації в Інтернеті.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

- Можливість взаємодії з системою шляхом REST запитів.
- Можливість загального моніторингу по заданому тексту.
- Можливість задавати критерії та діапазон для пошуку в залежності від потреб.
- Отримання результатів моніторингу на задану почту.
- Можливість системи паралельно оброблювати та аналізувати отримані результати.
- Можливість задавати сайт для моніторингу.

5.2. Вимоги до програмного забезпечення

- MS Windows XP/Vista/7/8/10 / Linux
- Віртуальная машина Java VM (Java SE Runtime Environment 8 і вище).
- Scala SDK 2.12 +
- Sbt 1.10 +

5.3. Вимоги до апаратної частини

- Комп’ютер з підключенням до мережі Інтернет

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	15.12.2019
Скадання і узгодження технічного завдання	25.01.2020
Створення модулів розроблюваної системи	20.02.2020
Тестування окремих модулів системи	15.03.2020
Доопрацювання, відладка і виправлення помилок	15.04.2020
Оформлення документації дипломної работи	19.05.2020

Пояснювальна записка
до дипломного проекту

на тему: «Веб-сервіс для моніторингу Інтернет-медіа ресурсів»

Київ – 2020

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

NLP (Natural Language Processing)	штучна обробка мови
NER (Naming entity recognition)	ідентифікація сутності
JVM (Java Virtual Machine)	віртуальна машина Java
HTML (HyperText Markup Language)	стандартизована мова розмітки документів
DOM (Document Object Model)	програмний інтерфейс , що дозволяє програмам і скриптам отримати доступ до вмісту
XML (Extensible Markup Language)	стандарт побудови мов розмітки
REST (Representational State Transfer)	підхід до архітектури мережевих протоколів
API (application programming interface)	Метод для взаємодії однієї системи з іншою
UML (Unified Modeling Language)	уніфікована мова моделювання

ВСТУП

Одною із основних потреб людського життя є інформація. Ми з кожним днем дізнаємось щось нове, читаєм, дивимось ТБ, слухаємо радіо, спілкуємося з людьми. Інколи людина, користуючись Інтернетом, втрачає багато свого дорогоцінного часу, щоб найти конкретну інформацію, адже потрібно переглянути не один сайт, наткнутися на рекламу, інтерфейс деяких сайтів є доволі таки складним і найти там щось потрібне займає багато часу. Інколи людина заходить в тупік, потрібна інформація не знайдена, залишається обмаль часу на пошук. Виникає потреба в пришвидшені процесу збору інформації по сайтах.

В даний час всі процеси, де застосовується синтаксичний аналіз, використовують парсери - програми для проведення візуального або програмно-автоматизованого синтаксичного і лексичного аналізу або розбору будь-якого документа з метою вилучення з нього необхідних даних. Проте лише одного парсера замало, потрібна система, яка б змогла ефективно застосовувати його для пошуку потрібної користувачу інформації та надавати результат своєї роботи – система моніторінгу контенту.

Для чого, наприклад, можна використовувати таку систему. По-перше, як відомо, найкращі сайти - це ті Інтернет-ресурси, на яких є цікава актуальна інформація. Нікому не потрібні вчорашні новини і сенсації, наприклад. Також, коли мова йде про сайти-обмінники валют, де необхідно змінювати інформацію про курс валют часом по кілька разів на день, створення моніторингу вкрай необхідно. Система в таких випадках буде виконувати всю роботу сама, цілодобово відстежуючи зміни курсу валют в Нацбанку. По-друге, така система необхідниа для автоматичного відстеження вашого Інтернет-ресурсу. Користувачі, які зайшли на вашу сторінку один раз і виявили там застарілу інформацію, більше ніколи не повернуться на неї. Саме тому для збереження постійних користувачів, а також для залучення нових необхідне регулярне оновлення інформації на

вашій інтернет-сторінці. По-третє, моніторинг медіа-ресурсів - ідеальний інструмент для того щоб не втрачати дорогоціний час на пошук тієї чи іншою інформації.

І, по-четверте, - централізація даних. Відомо, що інформації в мережі предостатньо, при цьому самої різної. Вся проблема в тому, що вся ця інформація розкидана по безлічі Інтернет-рессурсах і зібрати її не так просто. Використовуючи моніторинг різних сайтів, можна об'єднати вичерпну кількість корисної інформації в системі, тим самим залучаючи все більше відвідувачів.

Саме для вирішення такої задачі присвячена ця дипломна робота. Ми розробимо веб сервіс для моніторингу інтернет ресурсів, який буде шукати наявність потрібної користовачу інформації в найпопулярніших медіа ресурсах, фільтрувати серед неї найцікавішу за допомогою алгоритмів природної мови (nlp) і передавати її назад юзеру на електрону почту, також користувач сам зможе вказати певний медіа ресурс и система буде моніторити його для пошуку необхідної інформаціїї.

РОЗДІЛ 1
ОСНОВИ СИСТЕМИ МОНІТОРИНГУ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

В наш час великі масиви інформації повині підлягати обробці та аналізу. На основі фактів и припущень, отриманих з відкритих джерел, можливо не тільки аналізувати стан справ в тій чи іншій області, але й будувати прогнози розвитку ситуації загалом, що життєво важливо для прийняття потрібних рішень. Тож головною метою даної дипломної роботи є розробити універсальну, гнучку систему, яка буде мати змогу паралельно моніторити велику кількість ресурсів та оброблювати отриману з них інформацію. За основу такої системи візьмемо:

1. Модель акторів
2. Базові принципи парсингу сайтів
3. Логічний аналіз отриманої інформації за допомогою NER процесору

Розглянемо перші ці три пункти більш докладно.

1.1. Модель акторів

Модель акторів – математична модель обчислень, що розглядає «акторів» як універсальні примітиви паралельних обчислень. Актор – це обчислювальна сутність, яка може надсилати повідомлення тільки тим акторам, адреси яких знає. Після того як актор отримує повідомлення, він може одночасно: надсилати повідомлення на адреси інших акторів; створювати нових акторів; визначати, яким чином оброблювати наступні повідомлення. Визначальною властивістю моделі акторів є паралельність обчислень як усередині акторів, так і поміж ними, динамічне створення акторів, включення адрес акторів до повідомлень, взаємодія лише через прямі асинхронні повідомлення без будь-яких обмежень на порядок їх отримання.

Найпоширеніший варіант реалізації моделі запропоновано у 1985 р. У кожного актора є унікальне ім’я і поведінка, яка визначає його реакцію на отримані повідомлення. Щоб надіслати актору повідомлення,

використовується його ім’я. Коли у стані очікування актору надходить бажане повідомлення, він виконує обчислення, які визначені його поведінкою на випадок отримання саме цього повідомлення. Актор може виконувати три типи дій: відправляти повідомлення, створювати нових акторів і оновлювати свій локальний стан. Поведінка актора може бути змінена зі зміною локального стану. Актори не мають спільного стану: потрібно явно надіслати повідомлення до іншого актора, щоб вплинути на його поведінку. Кожен актор виконує свої обчислення одночасно (й асинхронно) з іншими акторами. Маршрут проходження повідомлення, як і затримки в мережі, не визначаються. Тому порядок прибуття повідомлень є невизначеним. Відмітимо і хорошу реалізацію визначальних семантичних характеристик стандартної моделі акторів: інкапсуляція стану та атомарне виконання методу у відповідь на повідомлення, прозорість у плануванні акторів і в доставці повідомлень та прозорість розташування, яка сприяє розподіленню обчислень та мобільності [6]. У роботі [5] їх називають: прозорість (fairness), мобільність (mobility), прозорість розташування (location transparency), інкапсуляція (encapsulation). Інкапсуляція в контексті програмування полягає в інкапсуляції стану акторів та безпечних повідомленнях. Актор не може напряму доступитися до внутрішнього стану іншого актора, а лише може впливати на його стан актора за допомогою відправки повідомлень. У разі порушення цієї властивості два актори одночасно можуть доступитися до критичної секції іншого актора (один напряму, інший через відправку повідомлення) і таким чином порушити програмну семантику.

Щодо безпечних повідомлень, то не має бути спільного стану (sharedstate) між акторами. Семантика передачі повідомлень – «за значенням», що може вимагати копіювання об’єктів повідомлень. Але в багатьох фреймворках (наприклад, на JVM) повідомлення несуть у собі посилання на їхній вміст, створюючи спільний стан між акторами.

Використання таких фреймворків вимагає від програмістів використовувати незмінні об’єкти або їхні копії. Під прозорістю, або прозорим плануванням, розуміють обов’язковість отримання повідомлення адресатом. Повідомлення може не дійти тільки в тому разі, якщо актор-адресат не є активним. Інша властивість прозорості полягає в тому, що актор має стати активним, інакше цей актор ніколи не зможе отримати повідомлення. Прозорість розташування акторів надає можливість розробникам програмувати без врахування фізичного розташування акторів. Прозоре йменування полегшує автоматичну міграцію в середовищі виконання або на інші вершини в мережі. Така мобільність дозволяє використовувати балансування навантажень та відмовостійкість [5]. Мобільність – це можливість переміщення обчислення між різними вузлами в мережі. Через підтримку інкапсуляції об’єктно-орієнтовані мови можуть підтримувати мобільність на рівні об’єктів, але всі стеки ниток, що виконуються, повинні отримати інформацію про віддалений об’єкт. Більше того, коли стеки ниток повинні отримати доступ до такого об’єкта, стек виконання має бути переміщений на віддалений вузол для завершення обчислень, а потім переміщений назад.

1.2. Базові принципи парсингу сайтів

Для того щоб отримати необхідну інформацію з сайту, її треба вилучити з HTML структури даної сторінки. Цей процес має багато методів та шляхів його застосування та має назву парсинг. і саме про це поговоримо більш детальніше.

Є кілька підходів до вилучення даних:

- Аналіз DOM дерева, використання XPath.
- Парсинг рядків.
- Використання регулярних виразів.
- XML парсинг.
- Візуальний підхід.

Розглянемо кожен підхід більш детально

1.2.1. Аналіз DOM дерева

Дуже важливою складовою даного методу є аналіз DOM самої сторінки. За допомогою нього, усі необхідні дані можна отримати за певним критерієм, наприклад атрибутом елемента дерева. Також слід зауважити, що до будь якої частини DOM дерева можна дістатися за допомогою певної ієрархії елементів цього дерева наприклад:

body -> p [10] -> a [1] -> текст посилання

або застосувати даний підхід для цілої низни елементів, наприклад [3]:

body -> links -> 5 елемент -> текст посилання

Переваги цього підходу:

1. Рівень складності і тип даних не мають значення
2. Застосовуючи лише шлях до певного елементу можна отримати його значення

Недоліки такого підходу:

1. Генерація DOM дерева для різних сайтів в залежності від платформи може відрізнятися, що ускладнює парсинг.
2. Такі парсери розраховані на короткочасний пошук даних, так як шлях до елементу може змінится.
3. DOM-шлях може бути складний і не завжди однозначний

Цей підхід можна використовувати разом з бібліотекою Microsoft.mshtml, яка, по суті. є core елементом в Internet Explorer.

1.2.2. XPath парсинг

Важливим кроком для удосконалення парсингу DOM дерева є використання XPath. Основна ідея даного підходу полягає у тому, щоб використовуючи певний синтаксис описувати шлях до елементу без необхідності обходу

повного DOM дерева за певним шляхом. Це добре використовується усіма відомими бібліотеками jQuery та HtmlAgilityPack, як бачимо на рис 1.1:

```
UriHtmlProcessor proc = new UriHtmlProcessor(new Uri("http://habrahabr.ru/new/page1/"));
proc.Initialize();

var links = from l in proc.Links
      where l.Class == "topic" && EndsWithInt(l.Href) == true
      select new ResultItem{
        Link = l.Href,
        TopicName = l.Text.ToWindows1251()
      };
```

Рис 1.1 – Використання XPath парсингу

1.2.3. Парсинг строк

Незважаючи на те, що цей підхід не можна застосовувати для написання серйозних парсерів, я про нього трохи розповім.

Можливо також таке, що дані відображаються за певними критеріями, наприклад значення параметрів стандартні, а змінюються лише їх характеристики. У такому випадку дані можуть бути отримані без аналізу DOM дерева. Використання набору методів для аналізу рядків іноді (частіше - простих шаблонних випадках) більш ефективний ніж аналіз DOM дерева або XPath.

1.2.4. Регулярні вирази і парсинг XML

Дуже часто, коли HTML повністю парсились за допомогою регулярних виразів. Це в корені невірний підхід, так як таким чином можна отримати більше проблем, ніж користі.

Регулярні вирази необхідно використовуватися тільки для отримання даних, які мають строгий формат - електронні адреси, телефони і т.д., в рідкісних випадках - адреси, шаблонні дані.

Ще одним неефективним підходом є розглядати HTML як XML дані. Причина в тому, що HTML рідко буває дійсним, тобто таким, що його можна розглядати як XML дані. Бібліотеки, які реалізували такий підхід, більше часу приділяли перетворенню HTML в XML і вже потім безпосередньо парсингу даних. Тому краще уникайте цей підхід.

1.2.5. Візуальний підхід

Даний метод обробки структури сайтів є експериментальним і знаходиться в стадії розвитку. Його ідея полягає у тому, щоб без використання програмної мови або API «налаштувати» систему для отримання потрібних даних будь-якої складності і вкладеності. Про чомусь схожому (правда застосовним в іншій області) - методи аналізу веб-сторінок на рівні інформаційних блоків, я вже писав. Думаю, що парсери майбутнього будуть саме візуальними.

Проблеми при парсінгу HTML даних - використання JavaScript / AJAX / асинхронних завантажень дуже ускладнюють написання парсеров; різні движки для рендеринга HTML можуть видавати різні DOM дерева (крім того, двигуни можуть мати баги, які потім впливають на результати роботи парсеров); великі обсяги даних вимагають писати розподілені парсери, що тягне за собою додаткові витрати на синхронізацію.

Не можна однозначно виділити підхід, який буде 100% може застосовуватись у всіх випадках, тому сучасні бібліотеки для парсинга HTML даних, як правило, комбінують, різні підходи. Наприклад, HtmlAgilityPack дозволяє аналізувати DOM дерево (використовувати XPath), а також з недавніх пір підтримується технологія Linq to XML. Data Extracting SDK використовує аналіз DOM дерева, містить набір додаткових методів для парсинга рядків, а аткже дозволяє використовувати технологію Linq для запитів в DOM моделі сторінки.

1.3 NER як одна с задач аналізу даних при моніторингу

Ми розглянули модель системи і як само отримувати необхідні дані, тепер поговоримо детальніше про те як само аналізувати отримані дані і повертати користувачу найцікавіші з них.

Завдання NER (Naming entity recognition) - виділити спан сутностей в тексті (спан - безперервний фрагмент тексту). Припустимо, є новинний текст, і ми хочемо виділити в ньому сутності (деякий заздалегідь зафіксований набір - наприклад, персони, локації, організації, дати і так далі). Завдання NER - зрозуміти, що фрагмент тексту «1 січня 1997 року" є датою, "Кофі Аннан" - персоною, а "ООН" - організацією

Що таке іменовані сутності? У першій, класичній постановці, яка була сформульована на конференції MUC-6 в 1995 році, це персони, локації і організації. З тих пір з'явилося кілька доступних корпусів, в кожному з яких свій набір іменованих сутностей. Зазвичай до персонам, локаціях і організаціям додаються нові типи сутностей. Найпоширеніші з них - числові (дати, грошові суми), а також сутності Misc (від miscellaneous - інші іменовані суті; приклад - iPhone 6).

Нехай у вас є якийсь текст (або набір текстів), і дані з нього потрібно ввести в базу даних (таблицю). Класичні іменовані суті можуть відповідати рядкам такої таблиці або ж служити змістом якихось осередків. Відповідно, щоб правильно заповнювати таблицю, потрібно перед цим виділити в тексті ті дані, які ви будете в неї вносити. Однак, є кілька причин, чому NER є однією з найпопулярніших завдань NLP. По-перше, витяг іменованих сутностей - це крок в сторону "розуміння" тексту. Це може як мати самостійну цінність, так і допомогти краще вирішувати інші завдання NLP.

Так, якщо ми знаємо, де в тексті виділені сутності, то ми можемо знайти важливі для якогось завдання фрагменти тексту. Наприклад, можемо виділити тільки ті абзаци, де зустрічаються суті якогось певного типу, а

потім працювати тільки з ними. Це допомагає в завданні моніторингу, тому що цей потужний механізм довзоляє аналізувати і надавати користувачу тільки потрібну й найактуальнішу для нього інформації.

Припустимо, по заданим вами критеріям моніторингу отримано декілька тем, тож такий підхід дозволяє фільтрувати й показувати вам лише насправді інформаційно насищений текст, де именованих сутностей найбільше, наприклад, або де є необхідні вам критерії, наприклад – локація.

Крім того, сутності - це жорсткі і надійні колокації, їх виділення може бути важливо для багатьох завдань. Припустимо, у вас є назва іменованої суті і, якою б вона не була, швидше за все, вона неперервна, і всі дії з нею потрібно здійснювати як з єдиним блоком. Наприклад, переводити назва суті в назву сутності. Ви хочете перевести «Магазин XXX» на французьку мову єдиним шматком, а не розбити на кілька не пов'язаних один з одним фрагментів. Уміння визначати колокації корисно і для багатьох інших завдань - наприклад, для синтаксичного парсинга.

Класичною складністю, яка заважає нам жити при вирішенні найрізноманітніших завдань NLP, є різного роду неоднозначності в мові. Наприклад, багатозначні слова й омоніми. Є й окремий вид омонімії, що має безпосереднє відношення до задачі NER - одним і тим же словом можуть називатися зовсім різні сутності. Наприклад, нехай у нас є слово "Вашингтон". Що це? Персона, місто, штат, назва магазину, ім'я собаки, об'єкта, щось ще? Щоб виділити цей фрагмент тексту, як конкретну сутність, треба враховувати дуже багато - локальний контекст (те, про що був попередній текст), глобальний контекст (знання про світ). Людина все це враховує, але навчити машину робити це непросто.

Друга складність - технічна, але не потрібно її недооцінювати. Як би ви не визначили сутність, швидше за все, виникнуть якісь прикордонні і непрості випадки - коли потрібно виділяти сутність, коли не потрібно, що включати в

спан суті, а що ні і т. п. (Звичайно, якщо наша сутність - це не щось слабо варіативне, типу емейла, а проте виділяти такі тривіальні суті зазвичай можна тривіальними методами - написати регулярний вираз і не думати ні про яке машинне навчання).

Незважаючи на те що сутності часто бувають багатослівними, зазвичай завдання NER зводиться до задачі класифікації на рівні токенів. Кожен токен відноситься до одного з декількох можливих класів. Є кілька стандартних способів зробити це, але самий загальний з них називається BIOES-схемою. Схема полягає в тому, щоб до мітки суті (наприклад, PER для персон або ORG для організацій) додати деякий префікс, який позначає позицію токена в спанні суті. Більш детально:

- B - від слова beginning - перший токен в спання суті, який складається з більше ніж 1 слова.
- I - від словами inside - це те, що знаходиться в середині.
- E - від слова ending, це останній токен суті, яка складається більше ніж з 1 елемента.
- S - single. Ми додаємо цей префікс, якщо сутність складається з одного слова.

Таким чином, до кожного типу сутності додаємо один з 4 можливих префіксів. Якщо токен не відноситься ні до якої сутності, він позначається спеціальною міткою, зазвичай має позначення OUT або O.

Наведемо приклад. Нехай у нас є текст "Карл Фрідріх Ієронім фон Мюнхгаузен народився в Боденвердер". Тут є одна багатослівна сутність - персона "Карл Фрідріх Ієронім фон Мюнхгаузен" і одна однослівна - локація "Боденвердер".

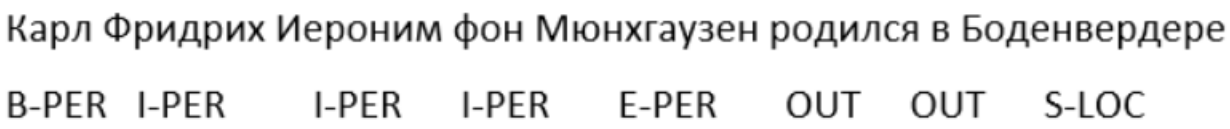


Рис. 1.2 – Приклад розкладу на сутності

Таким чином, BIOES - це спосіб відобразити проекції спаннів або анотацій на рівні токенів. Зрозуміло, що за такої розмітки ми однозначно можемо встановити межі всіх анотацій сутностей. Дійсно, про кожен токен ми знаємо, чи вірно, що сутність починається з цього токена або закінчується на ньому, а значить, закінчити чи анотацію суті на даному токені, або розширювати її на наступні маркери.

Переважна більшість дослідників використовує цей спосіб (або його варіації з меншою кількістю міток - BIOE або BIO), але у нього є кілька суттєвих недоліків. Головний з них полягає в тому, що схема не дозволяє працювати з вкладеними або пересічними сутностями. Наприклад, сутність "МГУ імені М.В. Ломоносова "- це одна організація. Але Ломоносов сам по собі - це персона, і це теж було б непогано поставити в розмітці. За допомогою описаного вище способу розмітки ми ніколи не зможемо передати обидва ці факти одночасно (бо у одного токена можемо зробити тільки одну позначку). Відповідно, токен "Ломоносова" може бути або частиною анотації організації, або частиною анотації персони, але ніколи не тим і іншим одночасно.

Крім стандартного способу звести задачу до класифікації на рівні токенів, є і стандартний формат даних, в якому зручно зберігати розмітку для завдання NER (а також для багатьох інших завдань NLP). Цей формат називається CoNLL-U.

Основна ідея формату така: зберігаємо дані в вигляді таблиці, де один рядок відповідає одному токену, а колонки - конкретного типу ознак токена (в т. Ч. Ознакою є і саме слово - словоформа). У вузькому сенсі формат CoNLL-U

задає, які саме типи ознак (колонки) включаються в таблицю - всього 10 типів ознак на кожен токен. Але дослідники зазвичай розглядають формат ширше і включають ті типи ознак, які потрібні для конкретного завдання і методи її вирішення.

1.4. Існуючі особливості системи контент моніторінгу медіа ресурсів та огляд існуючих рішень

Сама підготовка інформаційно-аналітичних матеріалів складається з ряду послідовних процедур, починаючи від впорядкування списку першоджерел для перегляду, розробки методик відбору та класифікації матеріалів, їх автоматичної обробки і закінчуючи аналізом занесеної в бази даних (БД) інформації і формуванням результатів моніторингу преси. У автоматизованої технології контент-моніторингу існує кілька важливих особливостей:

- використання ключового фрагмента публікації як одиниці формування
- текстового інформаційного масиву;
- формування банку ключових фрагментів публікацій є об'єднанням
- двох взаємопов'язаних автоматизованих процесів: аналітико-синтетичної
- переробки і багаторівневої процедури контент-аналізу текстів публікацій;
- індексація ключових фрагментів публікацій відбувається за допомогою

Унікальність запропонованої технології полягає в тому, щоб зробити об’єднання певних методів контент-аналізу, таких як кількісного і змістовного. Для контент-моніторингу використовуються монотематичний по наповненню системи з можливостями для багатоаспектного використання інформації при аналізі і підготовці матеріалів. Процедура контент-аналізу публікацій спрямована на виділення з тексту фрагментів, які відповідають найменшому, але цілісного модулю інформації в межах досліджуваної проблеми. В рамках такого модуля визначаються елементи проблеми, адекватні конкретним значенням класифікатора і між ними встановлюються зв'язки для подальшої формальної передачі змісту фрагмента публікації.

Інформація, яка не стосується проблеми, не виділяється з тексту і не заноситься в БД. Введений в інформаційну систему документ являє собою сукупність ключових фрагментів [17] тексту (кожен з яких заіндексувати відповідно до його змістом). Процедура обробки публікацій - своєрідне інформаційне сито, яке пропускає лише релевантні темі інформацію у вигляді фрагментів тексту. Серед переваг даної технології, слід відзначити зведення до мінімуму інформаційного шуму. Сформовані БД зберігаючи текст оригіналу, - досить компактні і зручні в роботі. серед недоліків технології - великі витрати інтелектуальної роботи як при обробці першоджерела, так і при наповненні БД. Спеціально написані програмні засоби розбивають вхідний документ на окремі незалежні фрагменти (інформаційні модулі), автоматично забезпечуючи кожен з них посиланням на бібліографічну інформацію, яка була внесена при описі документа. Користувачі отримують максимально коротку, але повну і об'єктивну інформацію з проблеми, яка цікавить їх на конкретному етапі.

Таким чином, вдається вирішувати основні інформаційні проблеми нашого часу: відсівати інформаційний шум, забезпечувати інформаційний пошук і відбір релевантної інформації з досить значних за обсягом масивів документів і вносити в інформаційний потік необхідну структурованість і системність при збереженні його безперервності. Інформаційні технології дозволяють вводити нові елементи, необхідні в окремих випадках конкретним споживачам. Технічно це вирішується шляхом додавання в фасетну формулу нових фасет і відповідних їм сукупностей. Таким чином, в процесі розвитку проблеми поповнюється і модифікується початковий варіант класифікатора, і, відповідно, вдосконалюється сама система в плані можливостей аналізу проблеми. Унікальність методології контент-моніторингу полягає також у тому, що вона не прив'язана ні до конкретної СУБД, ні до конкретних видів джерела інформації, ні до тематиці інформації. Хоча технології були більш ретельно випробовані на матеріалах сайтів соціально-політичного характеру, теоретично вони можуть бути використані

для моніторингу будь-яких інформаційних потоків, в тому числі і для інформаційних повідомлень в мережі інтернет. Розглянемо порівняльну характеристику існуючих сервісів на ринку с розроблюваним проектом у вигляді таблиці.

Таблиця 1.1 Порівняльна характеристика існуючих систем моніторингу

Назва проекту	Вартість послуг	Логічний пошук	Діапазон пошуку	Моніторинг заданих ресурсв
Brandwatch	Висока	Відсутній	Широкий	Присутній
Babkee	Середня	Відсутній	Середній	Відсутній
Awario	Середня	Присутній	Середній	Відсутній
Mention	Низька	Відсутній	Середній	Відсутній
Поточний проект	Безкоштовний	Присутній	Середній	Присутній

Розглянемо деякі порівняльні критерії. Під логічним пошуком мається на увазі можливість ресурсу не тільки знаходити необхідні дані в мережі, але й оброблювати її та сортувати по релевантності або іншим критеріям.

Діапазон пошуку:

- Широкий – основні соціальні мережі та засоби масової інформаціїї
- Середній – вибірковий спектр соціальних мереж та ЗМІ

Моніторинг заданих ресурсів – можливість надавати змогу користувачу моніторити власноруч заданий ресурс по заданим критеріям.

ВИСНОВКИ ДО РОЗДІЛУ 1

В ході виконання даного розділу була поставлена за мету реалізація системи моніторингу та показані теоретичні підходи її реалізації з використанням моделі акторів, парсингу сайту та NLP процесору. Також була проведена порівняльна характеристика існуючих рішень на ринку по системам моніторингу інтернет-медіа ресурсів. Що ж, в процесі розгляду було з'ясовано, що представлені аналоги мають ряд мінусів. Серед цих мінусів такі, як:

1) Надто об'ємні. Великі програми з поганою масштабованістю.
2) Не гнучкі. Надані системи не є надто гнучкими для усіх випадків життя. Тому іноді треба вдаватися до написання свого схожого програмного коду.
3) Це платні програми з поганим аналізом и критеріями пошуку інформації

З іншого боку, розроблювана система моніторингу інтернет медіа ресурсів якраз таки немає мати тих недоліків, що я описав вище.

Тому, саме через це я хочу розробити власну систему моніторингу, що буде вкрай гнучкою, матиме розширюваний функціонал, і буде так само гарно працювати як для великих, розподілених кластерних систем, так і для маленьких мобільних пристроїв.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ

2.1. Визначення вимог і завдання

До основних функцій системи можна віднести такі як:

1) Моніторинг заданого тексту по обраним критеріям
2) Моніторинг заданого тексту в загальному діапазоні пошуку
3) Моніторинг тексту по кількісному критерію
4) Можливість отримувати результати моніторингу по електроній почті
5) Моніторинг тексту з логічним пошуком
6) Можливість для користувача задавати джерело для моніторингу
7) Конфігурабельність частоти критеріїв для моніторингу

Основні вимоги до системи:

1) Асинхрона обробка та виконання моніторингу по усьому спектру компонентів
2) Відображення можливих помилок при моніторингу
3) Універсальний підхід парсингу для різних ресурсів
4) Невеликий розмір

2.2. Концептуальна діаграма структури роботи системи моніторингу інтернет-медіа ресурсів

На основі необхідного функціоналу і вимог які окреслені вище, побудуємо концептуально-структурну діаграму роботи моделі (рис 2.1) даної системи моніторингу інтернет-медіа ресурсів. Використаємо патерн (шаблон) проектування Посередник (англ. Mediator) для покращення зручності роботи з структурами. Посередник - це поведінковий патерн проектування, який дозволяє зменшити зв'язаність безлічі класів між собою, завдяки переміщенню цих зв'язків в один клас-посередник.

Патерн посередник змушує об'єкти спілкуватися не безпосередньо один з одним, а через окремий об'єкт-посередник, який знає, кому потрібно

перенаправити той чи інший запит. Завдяки цьому, компоненти системи будуть залежати тільки від посередника, а не від десятків інших компонентів. У нашому прикладі посередником буде ActorDispatcher.

В нашій системі, як бачимо з діаграми 2.1, головну роль конфігурації моніторингу та передачі необхідних даних з RestController-у (який відповідає за отримання http реквесту від користувача) до усіх інших акторів, які власне і виконують необхідну роботу по парсингу сайту, аналізу отриманої інформації та надання результату моніторингу для головних компонентів системи, таких як DispatcherActor та RestController. Як я вже казав, головна конфігурація системи, а саме частота моніторингу та кому й коли віддавати сигнали на обробку даних, також виконує DispatcherActor, який власне і є реалізатором паттерну посередник.

З діаграми видно, що усі актори з якими “спілкується” ActorDispatcher відповідають за знаходження інформації на різних інтернет-медіа ресурсах по своїм заданим критеріям, MusicMonitorActor, для моніторингу інформації стосовно музики, GamesMonitorActor – для інформаціх стосовно ігр, NewsMonitorActor – для інформації стосовно загальних новин і GeneralMonitorActor – для парсингу спеціальних сайтів заданих користувачем.

Варто також відмітити, що уся взаємодія ActorDispatcher-а з іншими акторами, як і було вказано у функціональних вимогах до системи, відбувається асинхроно.

Це дозволяє системі дуже гнучко та ефективно оброблювати запити користувача незважаючи на їх динаміку та кількість, так як усі вони будуть виконуватись у різних потоках паралельно.

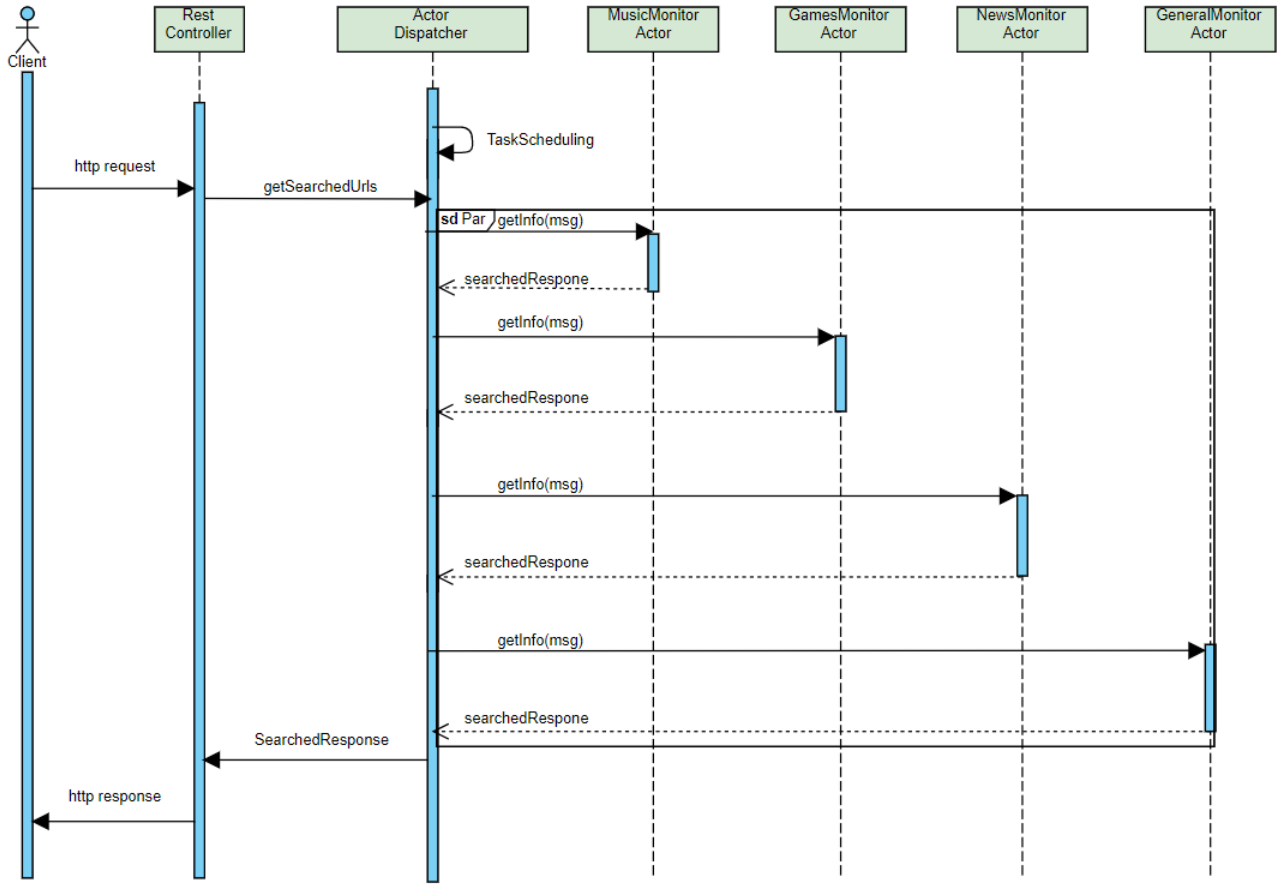


Рис. 2.1 – Концептуально-функціональна діаграма системи

2.3. Визначення прецедентів використання

Після того, як ми визначили функції, які повинна реалізувати система моніторингу інтернет-медіа ресурсів, розглянемо сценарії використання системи більш детально та побудуємо діаграму варіантів використання. На рис.2.1 зображена діаграма прецедентів високого рівня. Тут показані дії розробника-користувача, які абстраговані від деталей. Більш детальна ієрархія прецедентів зображена на рис. 2.2. – 2.6.

За допомомогою наведеної нище ієрархії прецедентів краще розкриваються способи застосування розроблюваної системи та наглядно видно як саме користувач може з нею взаємодіяти.

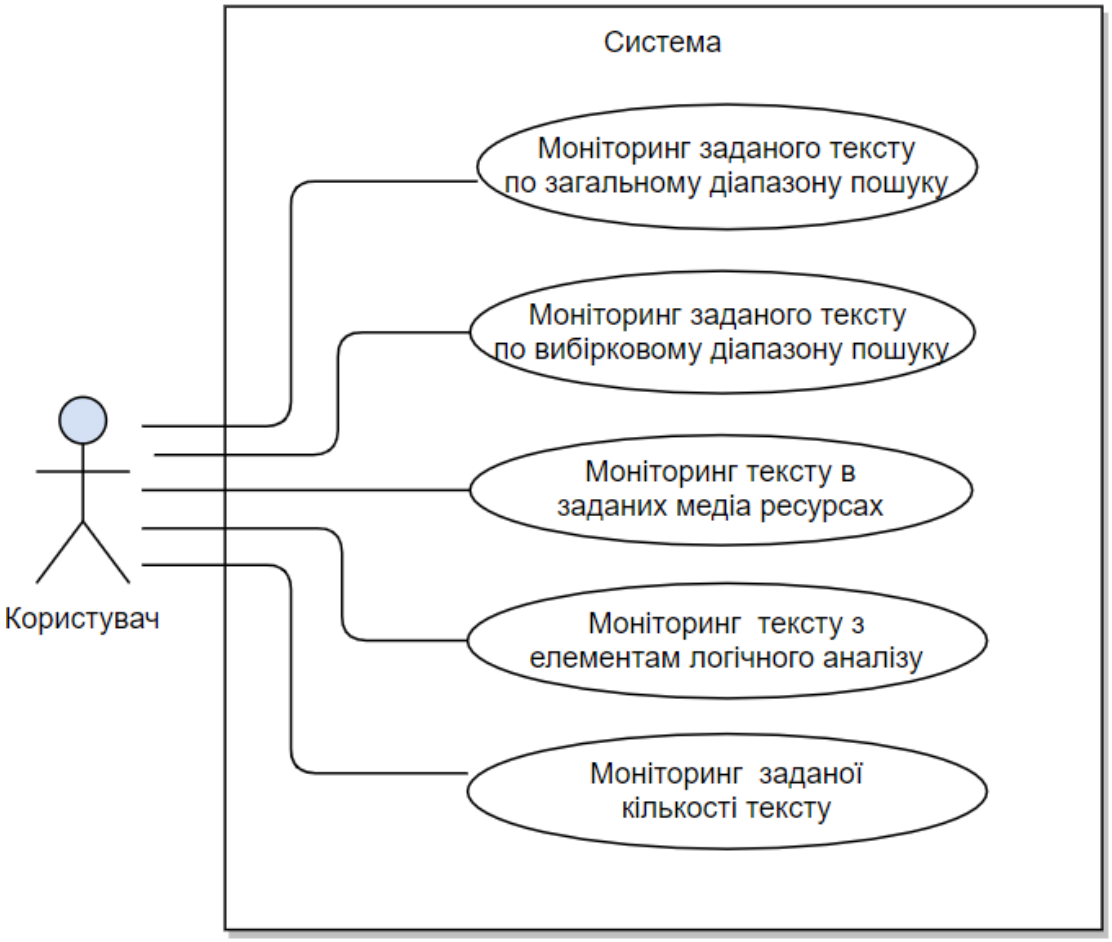


Рисунок 2.1 – Діаграма прецедентів додатку

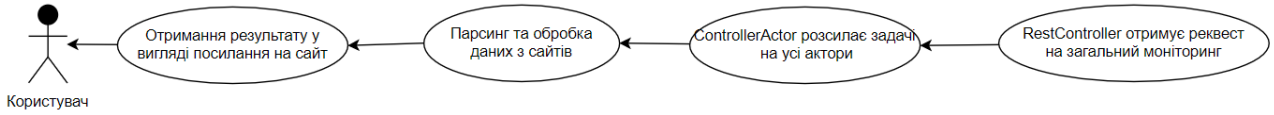


Рисунок 2.2 – Ієрархія прецеденту «Моніторинг заданого тексту по загальному діапазону пошуку»

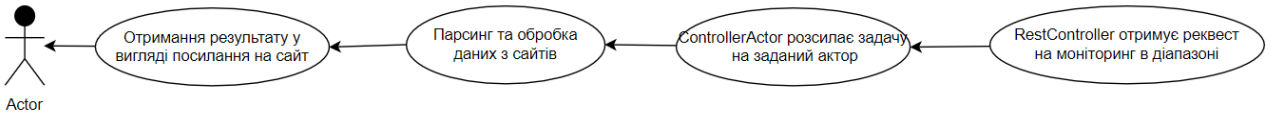


Рисунок 2.3 – Ієрархія прецеденту «Моніторинг заданого тексту по вибірковому діапазону пошуку»

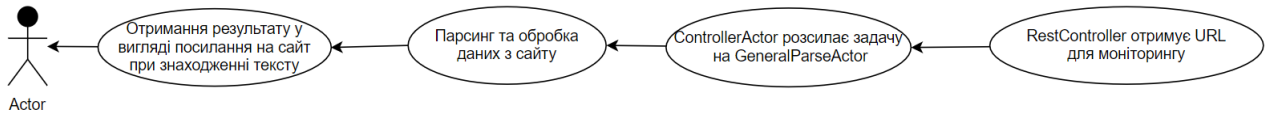


Рисунок 2.4 – Ієрархія прецеденту «Моніторинг тексту в заданих медіа ресурсах»

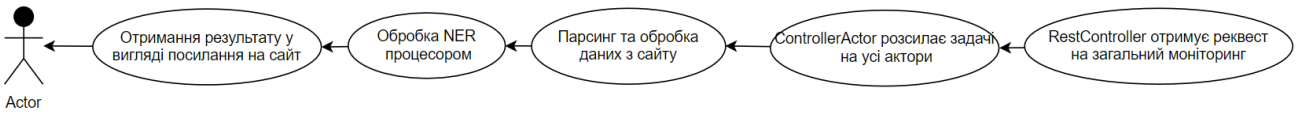


Рисунок 2.5 – Ієрархія прецеденту «Моніторинг тексту з елементам логічного аналізу»

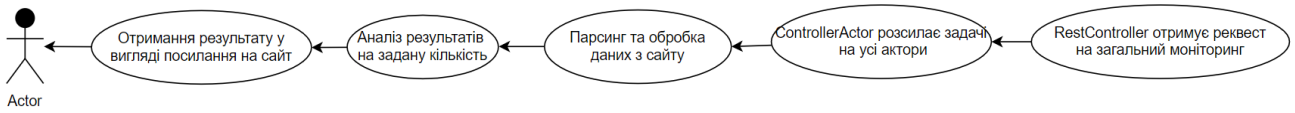


Рисунок 2.6 – Ієрархія прецеденту «Моніторинг заданої кількості тексту»

Давайте детальніше розглянемо такі прецеденти (англ. use-cases) як

1) Моніторинг заданого тексту по загальному діапазону пошуку;
2) Моніторинг заданого тексту по вибірковому діапазону пошуку;
3) Моніторинг тексту в заданих медіа ресурсах;
4) Моніторинг тексту з елементам логічного аналізу;
5) Моніторинг заданої кількості тексту

2.3.1. Моніторинг заданого тексту по загальному діапазону пошуку

Цей прецедент використовується користувачем для моніторингу необхідного йому текста по усім заданим пошуковим критеріям, а саме – музика/ігри/новини.

Розглянемо даний прецедент в табличці 2.1, що нище

Таблиця 2.1 Опис прецеденту «Опосередкована перевірка об’єкта»

Дії користувача	Відгук системи
1. Користувач робить виклик Rest API с заданим текстом для пошуку	*2. Система виконує паралельний пошук і обробку усіх акторів і повертає назви сайтів з потрібною користовачу інформацією у вигляді електроного письма.*

2.3.2.Моніторинг заданого тексту по вибірковому діапазону пошуку

Цей прецедент використовується користувачем для моніторингу необхідного йому текста по одному із критеріїв пошуку, а саме – музика/ігри/новини. Розглянемо даний прецедент в табличці 2.2, що нище:

Таблиця 2.2 Опис прецеденту «Моніторинг заданого тексту по вибірковому діапазону пошуку»

Дії користувача	Відгук системи
1. Користувач робить виклик Rest API с заданим текстом для пошуку та критерієм пошуки(музика/ігри/новини)	*2. Система виконує виклик потрібного актору, який виконує обробку і повертає назви сайтів з потрібною користувачу інформацією у вигляді електроного письма.*

2.3.3. Моніторинг тексту в заданих медіа ресурсах

Цей прецедент використовується користувачем для моніторингу необхідного йому інтернет медіа-ресурсу по заданому тексту. Розглянемо даний прецедент в табличці 2.3, що нище:

Таблиця 2.3 Опис прецеденту «Моніторинг тексту в заданих медіа ресурсах»

Дії користувача	Відгук системи
1. *Користувач робить виклик Rest API с заданим текстом та URL для моніторингу.*	2. *Система виконує виклик потрібного актору, який виконує обробку і повертає назву сайту у разі знайденої інформації у вигляді електроного письма.*

2.3.4. Моніторинг тексту з елементам логічного аналізу

Цей прецедент використовується користувачем для моніторингу необхідного йому інтернет медіа-ресурсу по заданому тексту з отриманням тільки інформативної та релевантної інформації по усім критеріям пошуку. Розглянемо даний прецедент в табличці 2.4, що нище:

Таблиця 2.4 Моніторинг тексту з елементам логічного аналізу

Дії користувача	Відгук системи
1. *Користувач робить виклик Rest API с заданим текстом.*	2. *Система виконує паралельний пошук і обробку усіх акторів, з подальшою обробкою NER процесором і повертає назви сайтів у порядку найінформативніших у вигляді електроного письма.*

2.3.5. Моніторинг заданої кількості тексту

Цей прецедент використовується користувачем для моніторингу необхідного йому інтернет медіа-ресурсу по заданому тексту. Розглянемо даний прецедент в табличці 2.5, що нище:

Таблиця 2.5 Опис прецеденту «Моніторинг заданої кількості тексту»

Дії користувача	Відгук системи
1. *Користувач робить виклик Rest API с заданим текстом та критерієм пошуку у вигляді кількості повторювань*	2. *Система виконує паралельний пошук і обробку усіх акторів, з фільтрацією по кількості входжень і повертає назви сайтів у вигляді електроного письма.*

Слід зауважити, що перед використанням наведених вище прецедентів користувач має змогу вказати за допомогу Rest API свій електроний адрес, куди власне буде отримувати результати роботи системи, та частоту моніторингу для системи, тобто як часто будуть аналізуватися інтернет-медіа ресурси на наявність потрібної інформації.

2.4. Мова програмування системи

За основу системи була використана мова програмування Scala. Scala - мова програмування, винайдена паном Мартіном Одерським та його дослідницькою командою у 2003 році.

Scala - це компілятор та багатопрофільна мова програмування, яка є компактною, швидкою та ефективною. Головною перевагою Scala є JVM (віртуальна машина Java). Код Scala спочатку компілюється компілятором Scala і генерується в байткод, який потім буде переданий у віртуальну машину Java для отримання результату.

Серед основних переваг мови:

- Scala здатна працювати з даними, які зберігаються в розподіленому вигляді. Вона отримує доступ до всіх доступних ресурсів і підтримує паралельну обробку даних.
- Scala підтримує незмінні дані та підтримує функції вищого порядку.

- Scala - це оновлена версія Java, яка була розроблена для усунення непотрібного коду. Вона підтримує кілька бібліотек та API, що дозволить програмісту досягти меншого часу.
- Scala підтримує конструкції декількох типів, що дозволяє програмісту легко працювати з обгортками / типами контейнерів.
- Scala є як функціональною мовою програмування, так і об'єктно-орієнтованою мовою програмування. Кожна змінна та значення, яке використовується в Scala, неявно зберігається як об'єкт за замовчуванням.
- Scala може підтримувати декілька мовних конструкцій без необхідності будь-яких розширень, бібліотек та API для певної мови (DSL).

2.5 Використання фреймворку Akka

Для реалізації моделі акторів описаної в першому розділі був використаний фреймворк – Akka. Що таке Akka і чому саме Akka? Akka - це фреймворк для двох мов, які працюють під JVM, Java і Scala. Це фреймворк, що надає готову середу для побудови високонавантажених систем, які ефективно використовують ресурси машини, і навіть набагато більше:

- кластеризація з коробки
- опис логіки потокової обробки даних
- побудова відмовостійких реактивних систем з коробки

Вийшов він з рантайму мови Scala. В 2006 у розробників мови виникла ідея вивести його в окреме рішення, в рамках рантайму він надавав засоби розробки багатопотокових додатків з використанням легких потоків. Власне легковагі потоки (потоки користувальницького рівня) самі по собі і є засіб вирішення проблеми ефективної утилізації ресурсів. І ось в 2010 виходить перша версія (а саме 0.5) фреймворка і зараз вони дісталися вже до 2.5.12 Окреслимо в чому саме полягають переваги цього фреймворку.

Запропоноване рішення, яке надає доступ до бізнес-функціоналу моніторингу, скажімо 10'000 користувачів, які в активному режимі

підключені до мого сервера і шлють в нього запити. Якщо вирішувати в класичній парадигмі архітектури веб-серверів (веб-сервер - не обов'язково готове рішення як Apache HTTP Server або NGINX, а ваше власне застосування, що має доступ, в класиці, по протоколу HTTP), то виглядати це буде наступним чином: запит користувача створить в потоці, який розбирає нові підключення, процес породження нового потоку з контекстом цього підключень (засоби многопотоковості в цьому підході нам надає операційна система, а це потоки рівня ядра).

Вартість породження такого потоку (пам'ять + процесорний час) досить велике, також діспетчиризувати (перемикати конкести, зганяти з процесора, так як виділений квант для цього потоку закінчився) це все складно.

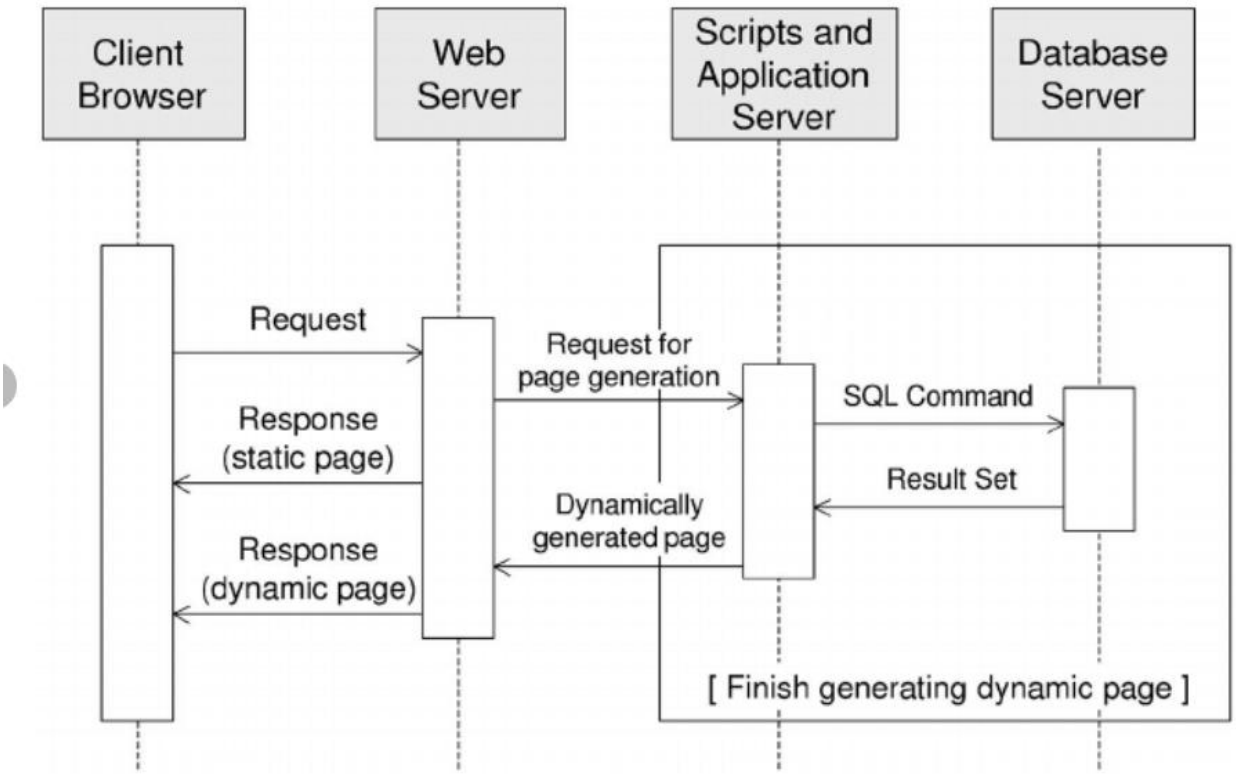


Рис. 2.1 – Класична архітектура веб-сервера

Тепер про те як Akka вирішує ці проблеми. І так, Akka - фреймворк, що надає засоби розробки високонавантажених систем. Робить він це,

використовуючи модель акторів, вражений мовою Erlang. Модель сама по собі дуже проста. Основна філософія - все є Актор.

А Актор - це сутність, яка приймає повідомлення, на які може відреагувати наступними діями:

- Надіслати N повідомлень інших акторів (крім себе)
- Породити M нових акторів
- Змінити внутрішній стан (що позначиться на наступні реакції на вхідні повідомлення)

Тепер як все це виглядає в Akka: Actor - клас описує логіку роботи з повідомленнями, успадкований від AbstractActor (UntypedActor, AbstractPersistentActor); по-суті, сутність, інкапсулюють логічно завершений функціонал, атомарно виконується в однопотоковому режимі (багатопотоковість досягається за рахунок породження кількох однакових екземплярів вашого актора).

Messages - екземпляр класу, відправлений одним актором асинхронно, і так само асинхронно отримується іншими.

Dispatcher - диспетчер, який відповідає за приміщення акторів на виконання на пулі потоків, коли приходить нове повідомлення для цього актора.

Mailboxes - черга повідомлень, що зберігає передані конкретному актору послання, в окремому випадку поштову скриньку ділиться групою акторів

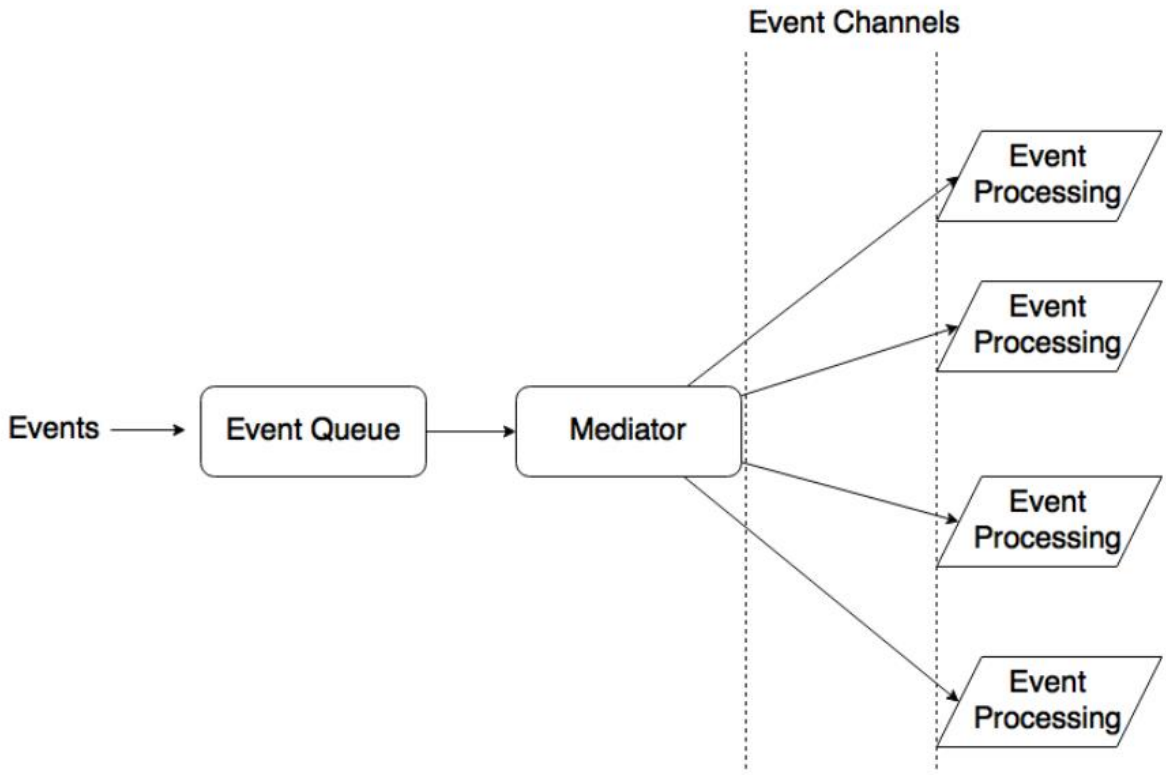


Рис. 2.4 – Асинхрона подійна система веб-серверу

Основні принципи, які слід пам'ятати:

- прямого звернення до екземпляру класу актора принципово немає, така архітектура, спілкування тільки через повідомлення
- не можна змінювати поля об'єкта повідомлення після відправки: середа ніяк це не контролює, середа вважає що повідомлення незмінне
- негарантована доставка; немає гарантії збереження послідовності повідомлень від різних акторів, але є гарантія послідовності від одного

Спочатку ми описуємо всіх учасників нашої системи:

- Акторів і повідомлення, які вони розуміють
- Наповнення акторів: то як вони повинні реагувати на відомі повідомлення (а так само на невідомі)
- Процес бутстрапінга всієї нашої системи (в рамках однієї програми можна створити кілька систем)

І все починається з першого повідомлення, яке потрапляє і передається деякому актору (через ActorRef і то як його можна отримати), а далі:

1. Про нове повідомлення оповіщається Dispatcher, який керує даним актором
2. Dispatcher ставить обробник актора вхідних повідомлень (Receive) на потік з керованого ним пулу
3. До тих пір, поки актор виконує обробку поточного повідомлення, нові повідомлення накопичуються в його Mailbox.
4. І так далі, відповідно до вищеописаної моделі

Ця вся система базується на моделі акторів, яку більш детально ми і розглянемо.

2.6. Обробка даних для моніторингу

Отже, ми розглянули модель в якій буде виконуватись система, тепер більш детально розберемо як нам отримати потрібні нам дані з сайтів для їх моніторингу. У даному підрозділі розглянемо структуру веб-сторінок. Розглянемо три підходи для отримання даних: регулярні вирази, Scala Scapper і nsxa. І вкінці порівняємо ці методи по різним критеріям.

2.6.1. Три підходи обробки веб-сторінок

Тепер, коли ми вже ознайомились із структурою веб-сторінок, розглянемо по порядку три абсолютно різні підходи парсингу даних:

1. регулярні вирази;
2. Scala scrapper модуль;
3. nsxa модуль.

Для того, щоб спарсити область, використовуючи [7] регулярні вирази, продемонструємо приклад пошуку вмісту елемента <textarea>, в такий спосіб:

```
val regex = "<textarea.*>(.+)</textarea>".r
regex.findAllIn(html).map {
  match => // process match
}
```

Регулярні вирази важко побудувати, так як вони нечитабельні. Крім того, є ще й інші незначні нюанси, які порушили б роботу програми, наприклад, якщо до тегу <textarea> розробники додадуть атрибут title, і на цьому наша функція вже не буде викновутись, потрібно заново переписувати регулярний вираз. З цього прикладу видно, що регулярні вирази забезпечують швидкий спосіб відокремлення даних, але дуже крихкі і легко ламаються, коли веб-сторінка оновлюється. На щастя, є більш ефективні рішення. **Scala Scrapper** є популярним модулем, який аналізує веб-сторінки, а також надає зручний інтерфейс для навігації контенту. Остання версія може бути встановлена за допомогою збірника проекту:

```
libraryDependencies += "net.ruippeixotog" %% "scala-scraper" %
"2.2.0"
```

Це бібліотека, що пропонує DSL для завантаження та вилучення вмісту зі сторінок HTML. Реалізація ознаки браузера, наприклад JsoupBrowser, може використовуватися для отримання HTML з Інтернету або для аналізу локального HTML-файла чи рядка:

```
    import net.ruippeixotog.scalascraper.browser.JsoupBrowser

    val browser = JsoupBrowser()

    val doc =
browser.parseFile("core/src/test/resources/example.html")

      val doc2 = browser.get("http://example.com")
```

Отриманий об’єкт - це документ, в якому вже передбачено кілька методів для маніпулювання та запиту HTML елементів. Для простих випадків використання цього може бути достатньо. Для інших ця бібліотека покращує процес вилучення вмісту, надаючи потужний DSL. Потім вміст може бути вилучений за допомогою оператора вилучення >> та запитів CSS:

```
// Отримати текст під айди тегом "header"
   doc >> text("#header")
```

В порівняні з регулярними виразами даний метод обробки є більш простим в застосуванній через вбудований DSL механізм і наявність простих операторів парсингу.

NSXA це скала модуль для парсингу xml структур, наприклад як в нашому випадку HTML. Насправді всі знають, що таке XML: це структурований, машиночитаний текстовий формат для представлення інформації, який легко перевірити на предмет «граматичності» тегів, атрибутів та їх зв’язку один з одним (наприклад, за допомогою DTD). Це контрастує з HTML, який може містити елементи, які не закриваються (наприклад, foo bar, а не foo bar) і все ще обробляються. XML лише коли-небудь мав бути форматом для машин, але він перетворився на представлення даних, яке багато людей закінчили (на жаль, для них) редагувати вручну. Однак навіть у машинозчитавоному форматі у нього є проблеми, такі як набагато більш багатослівний, ніж це дійсно потрібно, що має велике значення, коли вам потрібно перенести багато даних з машини на машину. Незважаючи на те, що існує багато застарілих даних, які були створені як XML, і існує багато спільнот (наприклад, спільнота цифрових гуманітарних наук), які все ще, схоже, обожнюють XML, тому люди, які виконують будь-яку розумну кількість роботи з аналізу тексту, швидше за все виявлять необхідність робота з кодованими XML даними.

Перше, що потрібно знати про XML у Scala, це те, що Scala може обробляти XML-літерали. Тобто вам не потрібно ставити лапки навколо рядків XML - натомість ви можете просто записати їх безпосередньо, і Scala автоматично інтерпретуватиме їх як XML-елементи (типу scala.xml.Element).

```
scala> val foo = <foo><bar type="greet">hi</bar><bar
type="count">1</bar><bar type="color">yellow</bar></foo>

foo: scala.xml.Elem = <foo><bar type="greet">hi</bar><bar
type="count">1</bar><bar type="color">yellow</bar></foo>
```

В усьому вищесказаному ми використовували XML-літерали - тобто вирази, введені безпосередньо у Scala, що інтерпретує їх як типи XML. Однак нам зазвичай потрібно обробити XML, який зберігається у файлі або рядку, тому об’єкт scala.xml.XML має кілька методів створення scala.xml.Elem об'єктів з інших джерел. Наприклад, наступне дозволяє нам створювати XML з рядка.

```
scala> val fooString = """<foo><bar type="greet">hi</bar><bar
type="count">1</bar><bar type="color">yellow</bar></foo>"""

fooString: java.lang.String = <foo><bar
type="greet">hi</bar><bar type="count">1</bar><bar
type="color">yellow</bar></foo>

scala> val fooElemFromString =
scala.xml.XML.loadString(fooString)

fooElemFromString: scala.xml.Elem = <foo><bar
type="greet">hi</bar><bar type="count">1</bar><bar
type="color">yellow</bar></foo>
```

Як бачимо працювати с HTML за допомогою scala xml також доволі просто та зрозумло. Проте для того щоб робити це ефективно потрібно заздалегідь знати структуру html файла, що не завжди є можливим. Порівняємо усі три методи і зробимо висновки:

Таблиця 2.6 Порівняльна характеристика методів парсингу

Підхід	Швидкодія	Простота у використанні	Простота в установці
regex	Висока	Складно	Легко
Scala Scrapper	Низька	Легко	Складно
nsxa	Висока	Легко	Легко

Отже, якщо потрібно спарсити невелекі об’єми даних, можна сміло використовувавти Scala Scrapper модуль. Або використовувати регулярні вирази, якщо потрібно вказати більш розширену унікальність критеріїв відбору даних. В загальному nsxa – це найкращий вибір для розробки автоматизованих систем парсингу, тому що він швидкий і надійний, а також не є складним у використанні.

2.7 NER як одна с задач аналізу даних при моніторингу

Для віришення NEP задачі в цьому проекті був використаний і кастомізований фреймворк Odin (Open Domain INformer) Структура вилучення подій (EE), заснована на правилах, називається Odin (Open Domain INformer) в пакеті org.clulab.odin.[15]

Цей фреймворк надає:

- Два повноцінні аналізатори дискурсу з теорією риторичної структури (RST). Аналізатори дискурсу прозоро включаються до наших процесорів природної мови (NL) (див. Нижче). Версія в CoreNLPProcessor покладається на складовий синтаксис, тоді як у FastNLPProcessor використовується синтаксис залежності. Вони виконують приблизно однаково, але останні набагато швидше.
- Пакет машинного навчання (ML) (org.clulab.learning), який включає реалізацію для загальних алгоритмів ML (наприклад, Perceptron, Logistic Regression, Support Vector Machines, Random Forests) як для класифікації, так і для ранжирування.

Набір процесорів NL в пакеті org.clulab.processors. Наразі ми надаємо такі API:

- CoreNLPProcessor - обгортка для CoreNLP Стенфорда, використовуючи її складовий аналізатор або аналізатор залежності;
- FastNLPProcessor - обгортка для CoreNLP Стенфорда, але використовуючи її аналізатор залежності нейронної мережі;
- BioNLPProcessor - версія CoreNLPProcessor, налаштована на біомедичний домен: краща токенізація для біомедичних текстів, покращене тегування POS для біодомену та користувацька NER для цього домену, яка розпізнає суб'єкти, що мають відношення до цієї галузі, такі як білки, хімічні та біологічні процеси;
- FastBioNLPProcessor - версія FastNLPProcessor, налаштована на біомедичний домен, подібно до BioNLPProcessor;
- CluProcessor - внутрішній процесор (ліцензований за ліцензією Apache), який містить: токенізацію (за допомогою Antlr), лематизацію (за допомогою MorphaStemmer), POS-тегів, розпізнавання названих об'єктів (NER) та неглибокий синтаксис чи ченкінг.

Останні три компоненти реалізовані за допомогою багатозадачної системи навчання, реалізованої за допомогою Scala обгортки DyNet. Продуктивність порівнянна з FastNLPProcessor, за більш дозвільною ліцензією. Крім того, слід пам'яті CluProcessor менший, ніж у FastNLPProcessor, тому він може бути більш підходящим для старих машин. Незабаром: аналіз залежності та підтримка декількох мов.

Процесори містять пакет машинного навчання (ML) (org.clulab.learning), який включає реалізацію для загальних алгоритмів ML (наприклад, Perceptron, Logistic Regression, Support Vector Machines, Random Forests) як для класифікації, так і для ранжирування.

Аналогічна структура існує для проблем з ранжуванням: RankingDataset використовується для зберігання корпусу прикладів ранжування, а RankingClassifier як API, який повинен бути реалізований усіма класифікаторами ранжирування.

ВИСНОВКИ ДО РОЗДІЛУ 2

1. Проаналізована модель для роботи з акторами та її реалізація у вигляді фреймворку Akka, розглянутий переваги цієї моделі для застосування в дипломному проекті.
2. В якості мови програмування була обрана мова Scala, оскільки вона підтримує багато корисних особливостей для роботи с парсерами та Akka.
3. Розглянуті та порівнянні основні методи парсингу та отриманий найефективніший з них – nsxa – робота з xml.
4. Розглянутий NER механізм та його застосування в данній дипломній роботі.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ МОНІТОРІНГУ

Ми розглянули основні вимоги до системи та розглянули модель акторів та основні технології за допомогою яких будемо виконувати розробку. Тепер в цьому розділі розглянемо саме проектування системи та опишемо як вона буде працювати та реалізовувати вимоги які були поставленні, а саме асинхроний моніторинг та планувальник для відладженого виконання роботи по парсингу сайтів.

3.1. Вибір допоміжних бібіліотек

Перш ніж перейти до структури проекту, ознайомимося з певними допоміжними бібліотеками, які допомагали в розробці. Для написання систем такого типу зазвичай використовують системи збірки та ряд певних бібліотек, які будуть допомагати в розробці. Вони значно допомогають та пришвидшують процес розробки та роблять так, щоб розробник мав змогу сконцентруватися на вирішені основної проблеми, саме ряд таких технологій ми розглянемо. У проекті використовуються наступні бібліотеки:

1) Система збірки проектів Sbt
2) Apache.commons-email
3) Scalatest
4) Cats-core

Перш за все розглянемо систему зборки за допомогою якої й можливе підключення усіх сторонніх бібліотек які допомогли в розробці.

Sbt – система програмних компонентів, яка використовуючи невелику кількість концепцій, пропонує гнучкі рішення збірки проектів. Дані рішення використовуються в розробці проектів для таких мов програмування як Java та Scala.

Завдання рівня конфігурації - це завдання, пов’язані з конфігурацією. Для цього розглянемо базові команди для роботи з системою. Наприклад,

compile, що еквівалентно ***compile:compile***, компілює основний вихідний код (конфігурація компіляції). ***test:compile*** компілює вихідний код тесту (конфігурація тестового тесту). Більшість завдань для конфігурації компіляції мають еквівалент тестової конфігурації, який можна виконати за допомогою ***test:*** prefix.

- ***compile*** Компілює основні джерела (у каталозі src / main / scala). тест: компілювати компільовані джерела тестування (у каталозі src / test / scala /).
- ***console*** Запускає інтерпретатора Scala класовим шляхом, включаючи зібрані джерела, всі банки в каталозі lib та керовані бібліотеки. Щоб повернутися до sbt, введіть: quit, Ctrl + D (Unix) або Ctrl + Z (Windows). Аналогічно тестуйте: консоль запускає інтерпретатор із класів тестів та classpath.
- ***consoleQuick*** Запускає інтерпретатора Scala із залежностей часу компіляції проекту від класного шляху. тест: consoleQuick використовує тестові залежності. Це завдання відрізняється від консольного тим, що не змушує складати джерела поточного проекту.
- ***consoleProject*** Вводить інтерактивний сеанс з sbt та визначенням збірки на classpath. Визначення збірки та пов'язані з ними значення пов'язані зі змінними, а загальні пакети та значення імпортуються. Додаткову інформацію див. У документації consoleProject.
- ***doc*** Створює документацію API для вихідних файлів Scala в src / main / scala, використовуючи scaladoc. тест: doc створює документацію API для вихідних файлів у src / test / scala.
- ***package*** Створює jar-файл, що містить файли у src / main / ресурси та класи, складені з src / main / scala. test: пакунок створює банку, що містить файли у src / test / ресурси та клас, складений з src / test / scala.
- ***packageDoc*** Створює файл jar, що містить документацію API, згенеровану з вихідних файлів Scala в src / main / scala. test: packageDoc створює банку,

що містить документацію API для файлів тестових джерел у src / test / scala.

- ***packageSrc:*** створює файл jar, що містить усі основні вихідні файли та ресурси. Упаковані шляхи відносяться до src / main / scala та src / main / ресурси. Аналогічно тестування: packageSrc працює над вихідними файлами та ресурсами тесту.
- ***run <argument> **** Запускає основний клас проекту на тій самій віртуальній машині, що і sbt. Основному класу передаються наведені аргументи. Будь ласка, перегляньте Запуск коду проекту для отримання детальної інформації про використання System.exit та багатопотокових читань (включаючи графічний інтерфейс) у коді, керованому цим дійством тест: запуск виконує основний клас тестового коду.
- ***runMain <main-class> <argument> **** Запускає вказаний основний клас для проекту на тій самій віртуальній машині, що і sbt. Основному класу передаються наведені аргументи. Будь ласка, перегляньте Запуск коду проекту для отримання детальної інформації про використання System.exit та багатопотокових читань (включаючи графічний інтерфейс) у коді, керованому цим дійством тест: runMain запускає вказаний основний клас у тестовому коді.

Це базові принципи для роботи з sbt, для реалізації системи моніторингу використовувався build.sbt файл де прописувалися усі допоміжні бібліотеки та технології які використовувалися для реалізації проекту. Так як сам файл та його конфігурація прописується на мові scala, то саме дана система збірки й була взята за основу, приклад с файлу build.sbt наведено нище:

```
lazy val akkaVersion = "2.5.22"
val akkaHttpVersion = "10.0.11"

libraryDependencies ++= Seq(
  "com.softwaremill.sttp.client" %% "core" % "2.0.0-RC9",
  "com.typesafe.akka" %% "akka-actor"      % akkaVersion,
```

```
  "com.typesafe.akka" %% "akka-actor-typed" % akkaVersion,
  "ch.qos.logback" % "logback-classic" % "1.2.3",
  "com.typesafe.akka" %% "akka-http"   % akkaHttpVersion,
  "com.typesafe.akka" %% "akka-http-spray-json" %
akkaHttpVersion,
  "com.typesafe.akka" %% "akka-stream" % akkaVersion,
  "com.typesafe.akka" %% "akka-actor-testkit-typed" %
akkaVersion % Test,
  "org.scalatest" %% "scalatest" % "3.0.8" % Test
)
```

Бачимо, що в зміну libraryDependencies додаються усі необхнідні допоміжні бібліотеки та задається їх версія та назва артифакту, який зберігається в загальному репозиторії. Для реалізації відправки електроного письма користувачу коли його запит буде знайдено, використана допоміжна бібліотека Apache common-emails. Commons Email має на меті створити API для надсилання електронної пошти. Він побудований поверх API Java Mail, який спрямований на спрощення. Деякі класи, які надаються, наступні:

1. **SimpleEmail** - Цей клас використовується для надсилання основних текстових електронних листів.
2. **MultiPartEmail** - Цей клас використовується для надсилання багаточастинних повідомлень. Це дозволяє текстове повідомлення з вкладеннями як вбудованим, так і вкладеним.
3. **HtmlEmail** - Цей клас використовується для надсилання електронних листів у форматі HTML. Він має всі можливості як MultiPartEmail, що дозволяє легко додавати вкладення. Він також підтримує вбудовані зображення.
4. **ImageHtmlEmail** - Цей клас використовується для надсилання електронних листів у форматі HTML із вбудованими зображеннями. Він має всі можливості як HtmlEmail, але перетворює всі посилання на зображення на вбудовані зображення.

5. **EmailAttachment** - це простий клас контейнерів, що дозволяє легко обробляти вкладення. Він призначений для використання з екземплярами MultiPartEmail і HtmlEmail.

Також для розробки системи були використані технології для тестингу - Scalatest, яка дозволяє за допомогою юніт тестування перевірити працездатність системи, та бібліотека cats, яка має на меті спростити процес розробки за допомогою використання патерну typeclasses.

3.2. Основні рішення з реалізації додатку та його компонентів

Розглянемо основні компоненти та рішення реалізації які були розроблені в даному дипломному проекті.

3.2.1. Компонентна архітектура проекту

Загалом структура проекту виглядає наступним чином – усі функціональні частини системи розбиті на окремі пакети, які логічно об’єднані за певними критеріями. Користувач взаємодія з системою за допомогою архітектурного стилю взаємодії компонентів під назвою REST. Користувач формую певний запит, а система на нього реагує, в нашому випадку використовуючи внутрішню модель акторів та методи парсингу сайтів. Білш детально взаємодію компонентів у системі та їх функціональність розглянемо далі. Ідея полягає в тому, щоб актори паралельно обмінювалися повідомленнями, в залежності від запитів користувача. Усі функціональні логічні блоки системи розбиті на пакети, які детальніше можна побачити на рисунку нище:

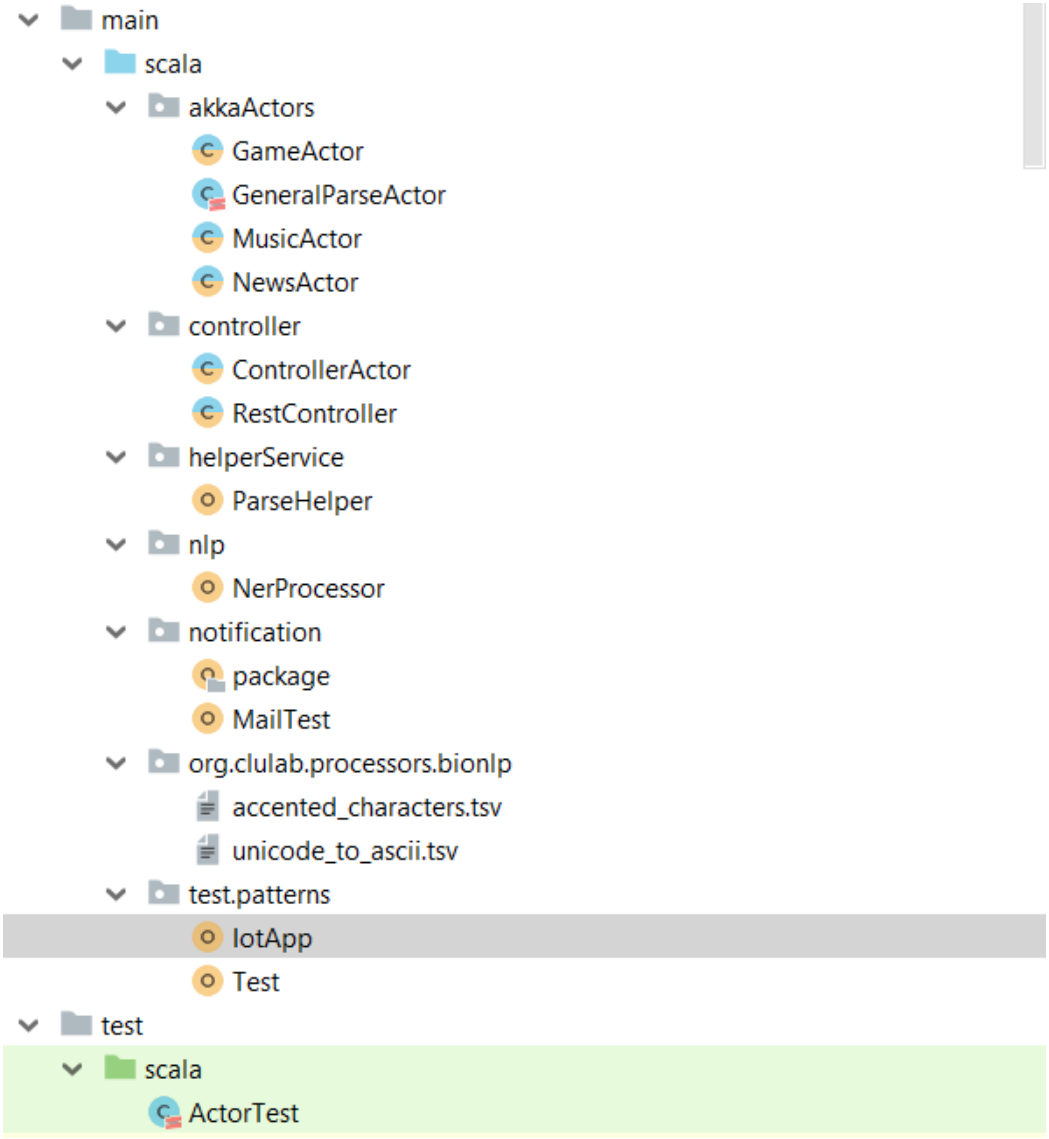


Рис 3.1 – пакетна структура проекту

Як бачимо компонентну архітектуру системи можна описати певними блоками. Для взаємодії з користувачем та управління усіма акторами використовується пакет controller, який має два важливих класи – RestController та ControllerActor, які ми детальніше розглянемо трохи згодом. Логічні блоки такі як відправка електроних листів, взаємодія з NER, актори для моніторингу – усе це відповідно також розділено на блок за пакетами, відповідно – notification, nlp, akkaActors. Детальніше кожен з цих блоків та їх взаємодію розглянемо в наступних розділах.

3.2.2. Взаємодія та аналіз компонентів системи

Тепер розглянемо та проаналізуємо компоненти системи та їх взаємодію один з одним на прикладі діаграми класів, наведеної нижче. На ній зображені основні компоненти системи та як саме вони працюють один з одним.

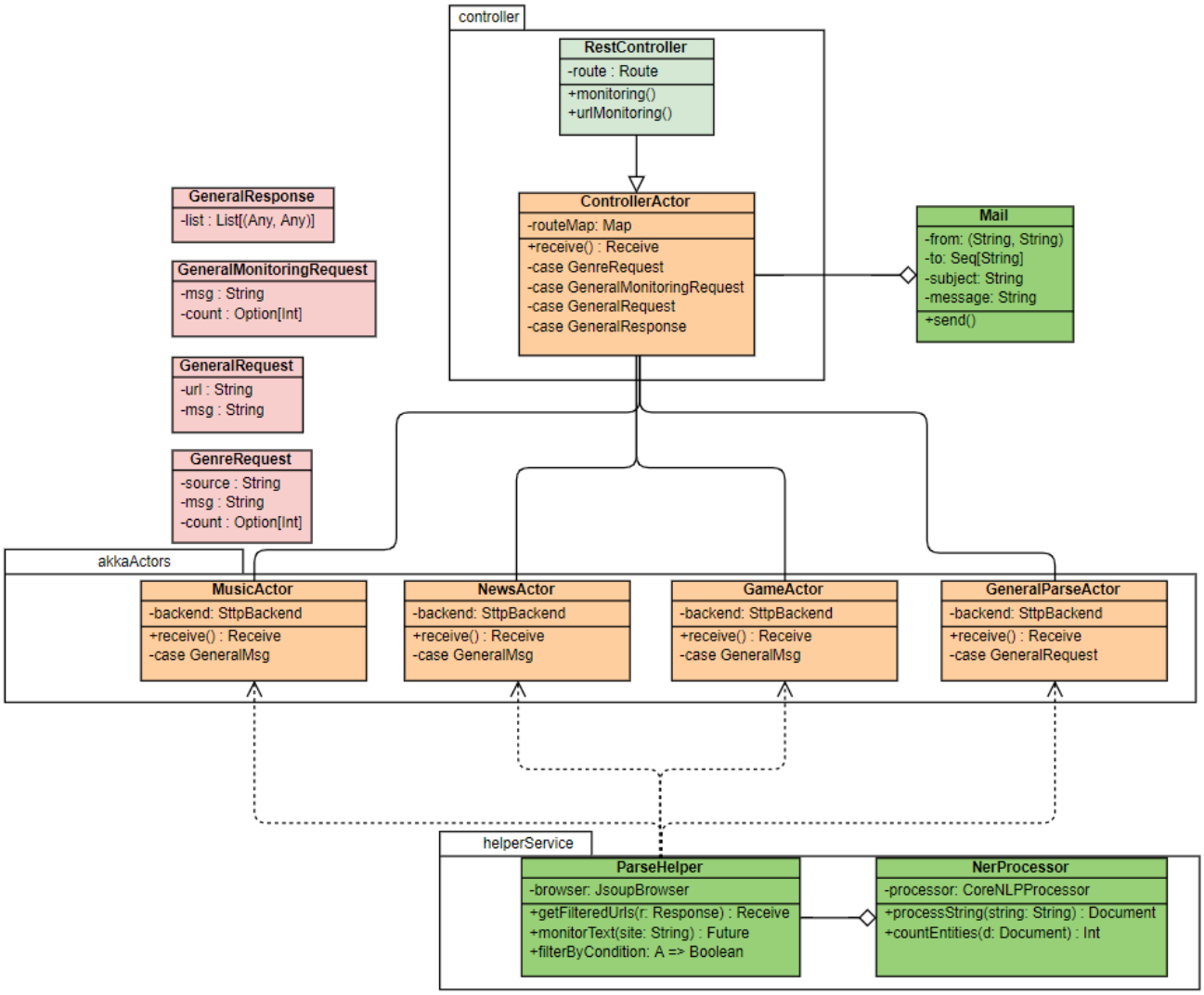


Рис. 3.2 – UML діаграма взаємодії компонентів системи

З діграми бачимо, що умовно систему можна поділити на три рівні – **component**, який отримує запит користувача, а саме класс RestController, і передає його до ControllerActor, який в свою чергу оброблює запит і направляє його до потрібного актора, де власне и починає свою роботу **akkaActors** компонент, який представляє собою збірку акторів, кожен з яких відповідає за пошук, аналіз та обробку інформації з відповідних ресурсів.

Цей компонент тісно пов’язаний з helperService, який в свою чергу надає певний спектр методів та функції для парсингу сайтів та аналізу отриманих результатів за допомогою NER процесору. Тож розглянемо кожен з цих компонентів більше детальніше та звернемо увагу на їх особливості

Controller

По-перше розглянемо головний клас який приймає реквест від користувача – RestController. Його особливість є також те, що він створює головний єкземпляр класс системи акторів – ActorSystem:

```
implicit val system = ActorSystem("monitoringSystem")
```

Сам класс використовує для реалізації рест архітектури бібліотеку Akka http.

HTTP модулі Akka реалізують повний стек HTTP на стороні сервера та клієнта поверх akka-act та akka-stream. Це не веб-рамки, а більш загальний набір інструментів для надання та користування послугами на базі HTTP. Незважаючи на те, що взаємодія з браузером, безумовно, також належить до сфери застосування, вона не є основним напрямком діяльності HTTP Akka.

Akka HTTP дотримується досить відкритого дизайну і багато разів пропонує декілька різних рівнів API, щоб «зробити те саме. Це означає, що якщо у вас виникли проблеми з використанням API високого рівня, є хороший шанс, що ви зможете зробити це за допомогою API низького рівня, який пропонує більшу гнучкість, але, можливо, вам потрібно буде написати більше коду програми. HTTP Akka орієнтувався на надання чіткої уваги на наданні інструментів для побудови інтеграційних шарів, а не ядер додатків. Як такий він вважає себе набором бібліотек, а не рамкою.

Однак якщо ваш додаток не є передусім веб-додатком, оскільки його основою є не взаємодія із браузером, а якась спеціалізована, можливо складна бізнес-послуга, і ви просто намагаєтесь підключити його до світу за допомогою інтерфейсу REST / HTTP, веб-рамки можуть не бути тим, що потрібно. У цьому випадку архітектура програми повинна диктуватися тим,

що для ядра має сенс не інтерфейсний шар. Крім того, ви, мабуть, не скористаєтесь можливо існуючими для браузера компонентами рамки, такими як шаблони перегляду, управління активами, генерація / маніпулювання / мінімізація / мінімізація / зміна JavaScript / CSS, підтримка локалізації, підтримка AJAX тощо.

У випадку розробки нашої моніторинг системи був ефективно використан базовий функціонал HTTP Akka для взаємодії з користувачем по REST-у. На рисунку нище показано як саме система приймає рест запит на моніторинг:

```
val route =
  path( pm = "monitoring") {
    get {
      parameters('msg.as[String], 'genre.as[String].?, 'count.as[Int].?) { (msg, genre, count) =>
       mainActor ! genre.map(value => GenreRequest(value, msg, count)).getOrElse(GeneralMonitoringRequest(msg, count))
        complete((StatusCodes.Accepted, "monitoring started"))
      }
    }
  }
```

Рис. 3.3 – Обробка запиту користувача

Другим не менш важливим класом в цьому комоненті є ControllerActor, головною задачою якого є власне запланувати весь моніторинг для акторів та керувати їм, також за допомогою цього класу обмін запитами виконується асинхроно, використовуючи механізми Akka. Саме планування виконується на основі екземпляру класу system наступним чином:

```
    system.scheduler.schedule(120 milliseconds, frequency,
generalParseActor, request)
```

Де frequency це частота моніторингу, яка задається користувачем. В залежності від запитів користувача які передає RestController, ControllerActor буде приймати рішення щодо того якому саме актору далі надати цей запит на обробку та аналіз.

akkaActors

Класи в цьому пакеті представляють собою набір акторів, яку виконують основну роботу по обробці і парсингу необхідних для них сайтів, які логічно розділені по жанрам і на які, в залежності від критеріїв пошуку заданих користувачем, відправляє запит ControllerActor. Надалі робота кожного актора проста – використовучи заданий набір сайтів і методи пакету helperService отримати необхідний набір сайтів який задовольняє критерії пошуку користувача і відправити їх назад до ControllerActor, який власне і виконає відправку електроною поштою. Окремої уваги варті два моменти, котрі я б і хотів розглянути детальніше. По-перше це те, як саме парситься заданий набір сайтів. І зараз мова не про сам процес парсингу, який ми розглянемо трохи згодом в наступному компоненті, а про те як це робиться ассинхроно за допомогою класу Future.

Future - це зручний спосіб обробляти кілька операцій, які виконуються паралельно. Це ефективний, не блокуючий спосіб. Загальна ідея проста: це Future як заповнювач для результату обчислення, який ще не існує. Загалом Future результати розраховуються паралельно і можуть бути узагальнені пізніше. Поєднання паралельних завдань таким способом часто призводить до швидшого, асинхронного, не блокуючого паралельного коду. За замовчуванням Future не блокує, використовуючи зворотні дзвінки замість типових операцій блокування. Future це об'єкт, який містить значення, які можуть бути доступні в якийсь момент. Ця величина часто є результатом якогось розрахунку. Оскільки обчислення може бути невдалим Future, може також утримувати виняток, якщо обчислення кидає помилку. Коли Future має значення або помилку, він вважається **завершеним** . Success означає, що Future завершилося вдало. Якщо завершено винятком , то результатом будет Failure. Future має важливу властивість, що його можна замінити лише один раз. Після того як об’єкт має значення або виняток, він по суті є незмінним і ніколи не перезаписується.

Тож розглянемо як Future допомогає у паралельній обробці сайтів у акторах на прикладі коду нище:

```
    Future.sequence(list.map(site => monitorText(site, find,
count))).map(s => s.flatten.flatten) onComplete {
  case Success(resultList) =>  mySender ! GeneralResponse(tuples
++ resultList)
  case Failure(exception) => println(exception)
}
```

Метод monitorText для кожного сайту створює об’єкт Future всередені якого виконується парсинг, і так як обробка не блокуюча, за допомогою методу sequence усі єкземпляри збираються в один і повертаються до відправника, а саме до ControllerActor, або виводиться повідомлення у разі виникнення помилки.

helperService

Даний пакет містить ParseHelper клас, методи якого використовуються акторами з akkaActors і допомагають в асинхроній обробці запитів користувача, парсингу та NER обробці отриманих ссилок. Розглянемо ці два методи більш детальніше.

`getFilteredUrls(response: Response[Either[String, String]], count: Option[Int], find: String)` – даний метод отримує результат парсингу та критерії пошуку від одного з акторів та оброблює його конвертуючи з типу Response до типу List, застосовуючи необхідні фільтрації. Також метод викликає NER процесор, який фільтрує оброблює отримані силки та фільтрує їх за кількістю так званих ‘іменних сутностей’, обробка виконується за допомогою наступного методу:

```
    titlesWithUrl.filter(filterByCondition).sortWith((a: Any, b:
Any) =>
  processString(a.asInstanceOf[String]) >
processString(b.asInstanceOf[String]) )
```

В результаті користувач буде отримувати силки відсортовані за інформативністю, яка отримана на основі обробки.

`monitorText(site: String, msg: String, count: Option[Int]): Future[Option[List[(Any, Any)]]]` – другий допоміжний метод, який обертає парсинг сайту в класс Future, який був розглянутий раніше, що дозволяє разпаралелити парсинг сайтів всередині актора, при тому що усі актори також виконують обробку при загальному моніторингу асинхроно.

3.3. Алгоритм роботи системи

Тепер коли ми розглянули компоненти роботи системи та їх взаємодію, можемо проаналізувати зашальний алгоритм за який буде функціонувати система, наведений як UML діаграма нище (Рис.3.4).

На діаграмі можемо побачити життєвий цикл запита користувача, який проходить три компоненти системи які були розглянуті вище. Починаючи з RestController який приймає запит та виконує його переадресацію, далі необхідне повідомлення отримую ControllerActor який виконує певну валідацію та в залежності від отриманого запиту посилає його одному з необхідних акторів. Це все відбувається за допомогою функції актора під назвою receive:

```
override def receive: Receive
```

Далі запит на обробку потрапляє в один з потрібних акторів, або в усі одночасно якщо користувач обрав загальний моніторинг. Актор аналізує запит і робить парсинг та пошук необхідного повідомлення за певними критеріями, такими як кількість слів, або лише загальний пошук за їх відсутності.

В результаті необхідні посилання які задовольняють критерії пошуку користувача повертаються до ControllerActor, який у свою чергу виконує відправку даних посилань за електроною поштою, яка була вказана. Діаграма алгоритму наведена нище.

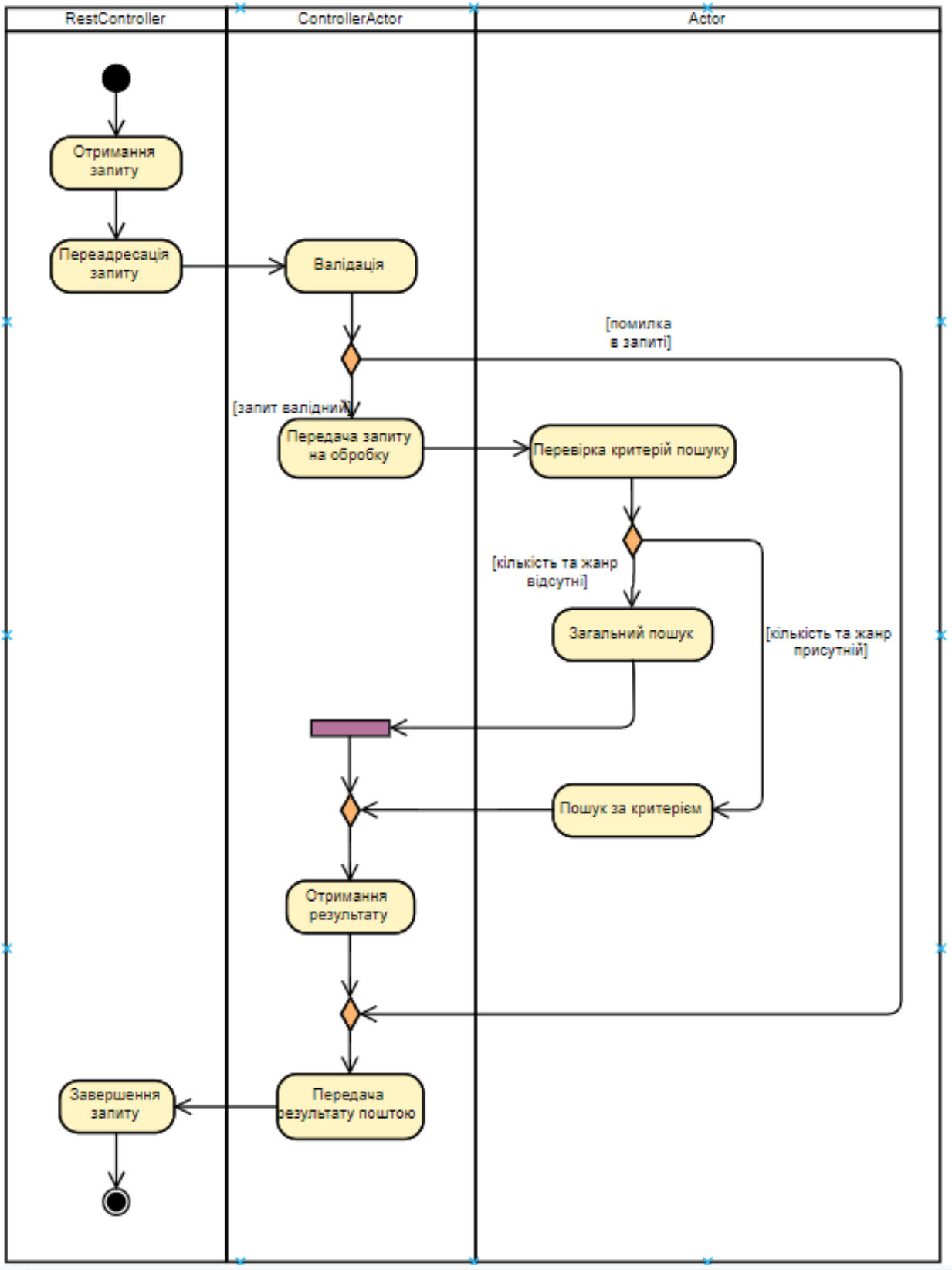


Рис 3.4 – Алгоритм-діаграма роботи системи моніторингу

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі були розглянуті допоміжні бібліотеки, які застосовувались при розробці системи моніторингу інтернет-медіа ресурсів та їх основне застосування. Також розглянуті основні компоненти системи та їх взаємодія один з одним.

В даному розділі також був побудований алгоритм роботи системи та розроблена діаграма взаємодії компонентів. Були описані основні структурні блоки системи та їх детальна реалізація.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ СИСТЕМИ МОНІТОРІНГУ

4.1. Тестування системи

Для тестування системи моніторингу була застосована бібліотека для покриття проекту юніт-тестами під назваою ScalaTest. Особливістю цієї бібліотеки є те, що вона дозволяє відтворити актор систему спеціально для тестингу, і покрити основні життеві цикли програми базовими тестами.

Юніт-тести - це рівень тестування програмного забезпечення, на якому тестуються окремі блоки / компоненти програмного забезпечення. Метою є перевірка виконання кожною одиницею програмного забезпечення відповідно до проекту. Пристрій - це найменша перевірена частина будь-якого програмного забезпечення. Зазвичай він має один або кілька входів і, як правило, один вихід. У процедурному програмуванні одиниця може бути індивідуальною програмою, функцією, процедурою тощо. У об'єктно-орієнтованому програмуванні найменша одиниця - це метод, який може належати до базового / надкласного, абстрактного класу або класу похідних / дочірніх. Рамки тестування блоків, драйвери, заглушки та макетні / підроблені об'єкти використовуються для допомоги в тестуванні одиниць.

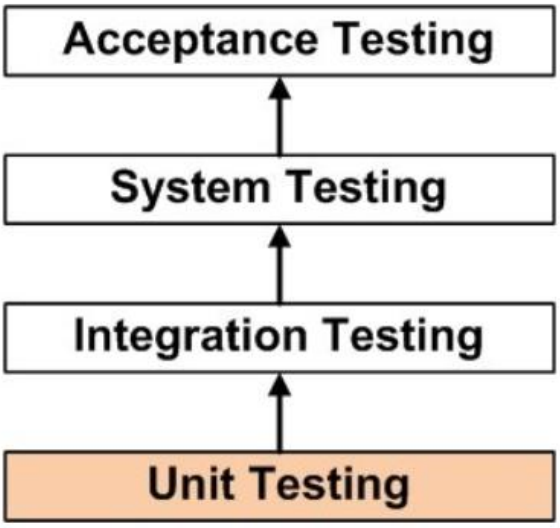


Рис. 4.1 – Діаграма тестування

ScalaTest також дозволяє писати тести використовуючи DSL, що значно полегшує їх розробку, приклад тесту та результати юніт-тестування системи наведені нижче:

```
    class RoutingSLipPattern extends
TestKit(ActorSystem("testsystem"))
  with WordSpecLike
  with ImplicitSender
  with StopSystemAfterAll {

  "Should send msg for monitoring" in {
    val probe = TestProbe()
    val router = system.actorOf(
      Props(new SlipRouter(probe.ref)), "SlipRouter")
    val minimalOrder = GeneralMsa(Seq())
    router ! minimalOrder
    val defaultMsg = GeneralResponse(
      list = List("test",”test”
      )
    probe.expectMsg(defaultMsg)
  }
```

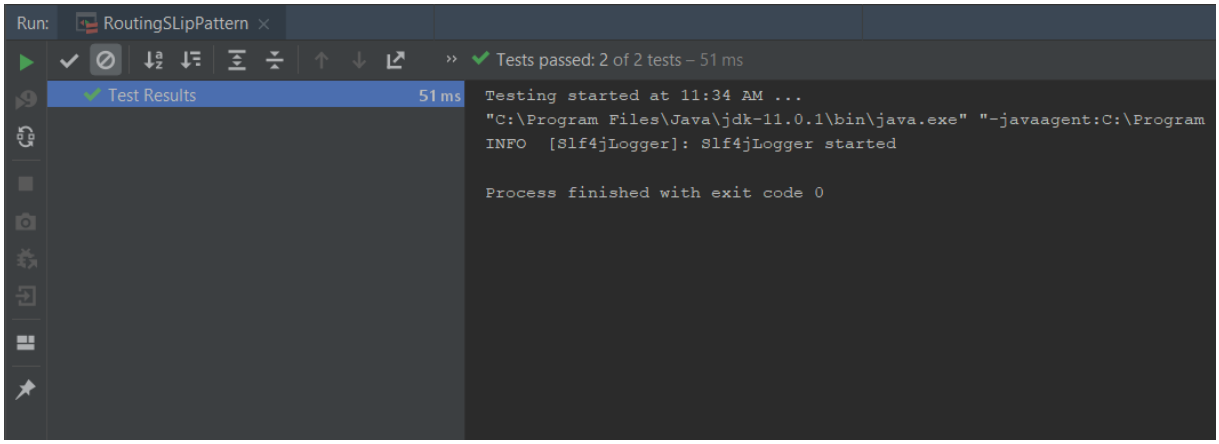


Рис. 4.2 – Результати юніт-тестування системи

4.2 Результати роботи системи

Так як розроблена система має REST-архітектуру, для взаємодії з нею використовуємо програму для посилання http запитів, наприклад postman. Для роботи з системою маємо два рест запити зі зміною кількістю параметрів:

GET -

`http://localhost:8080/monitoring?msg=been&count=1&genre=Games`

Де обов'язковим параметром є msg, який вказує на те що саме потрібно шукати для моніторингу, і два не обов'язкових параметри count і genre, які задають критерії пошуку. Якщо їх не вказати, пошук буде по загальним критеріям. Продемонструємо результати роботи:

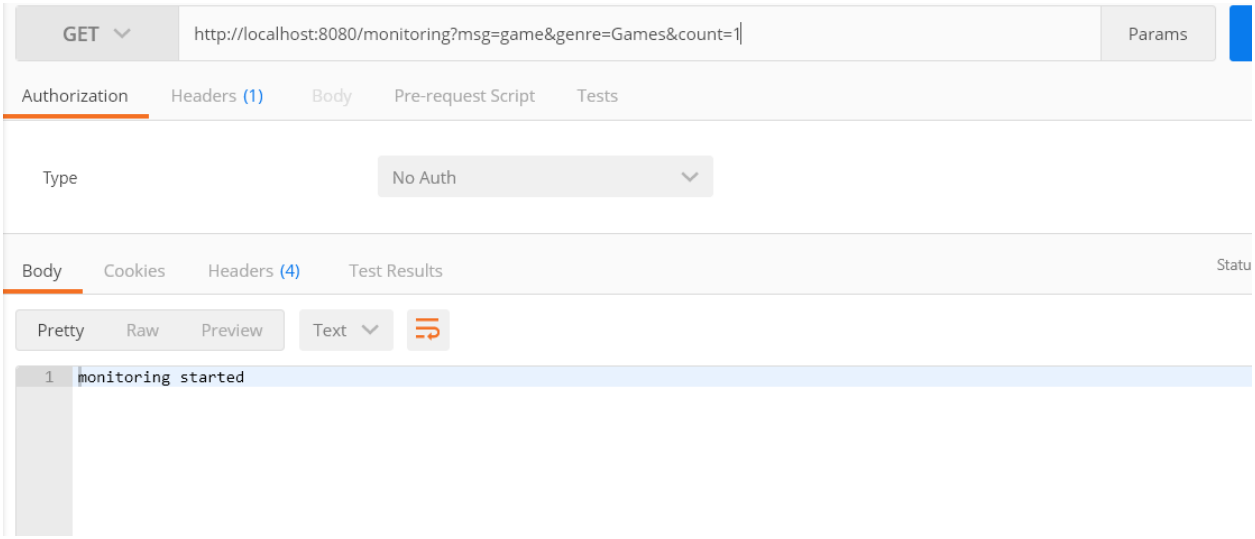


Рис. 4.3 – Запит на моніторинг

І отримуємо необхідні результати, які будуть надіслані користувачу за допомогою наперед заданої електроної почти, електроним листом:

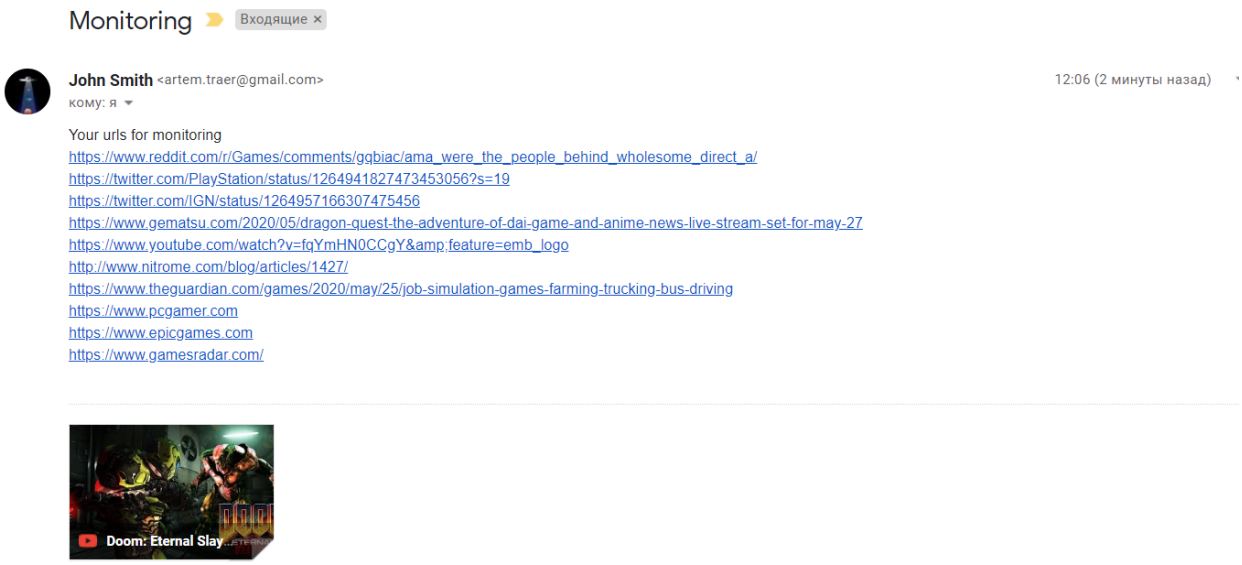


Рис. 4.4 – результати роботи системи

Також у користувача є можливість задати власну силку для моніторингу по заданому критерію, для цього використовується наступний запит:

POST - `http://localhost:8080/urlMonitoring`

```
{
   "url":"https://doc.akka.io/docs/akka-http/current/routing-dsl/overview.html?language=scala",
   "msg": "bindingFuture"
}
```

Задане посилання буде моніторитись і вам прийде повідомлення на задану пошту коли критерія вибірки буде задовільною.

4.2. Деплоймент системи

Ми будемо використовувати кластер Akka всередині контейнерів Docker. Потрібно бути особливо обережним з мережевою конфігурацією під час використання Docker. Щоб JVM добре працював у контейнері Docker, є деякі

загальні (не конкретні для Akka) параметри, які можуть потребувати налаштування. Розглянемо можливість використання

```
-XX: + UnlockExperimentalVMOptions -XX: +
UseCGroupMemoryLimitForHeap
```

параметрів для нашого JVM, що дозволяє зрозуміти обмеження пам'яті с-групи. Для багатопотокових програм, таких як JVM, обмеження планувальника CFS є невідповідним, оскільки вони обмежують дозволене використання ЦП навіть тоді, коли в хост-системі доступно більше циклів процесора. Це означає, що наша програма може втратити час процесора, але загалом система не працює. Нище наведена діагрма деплою для нашої системи:

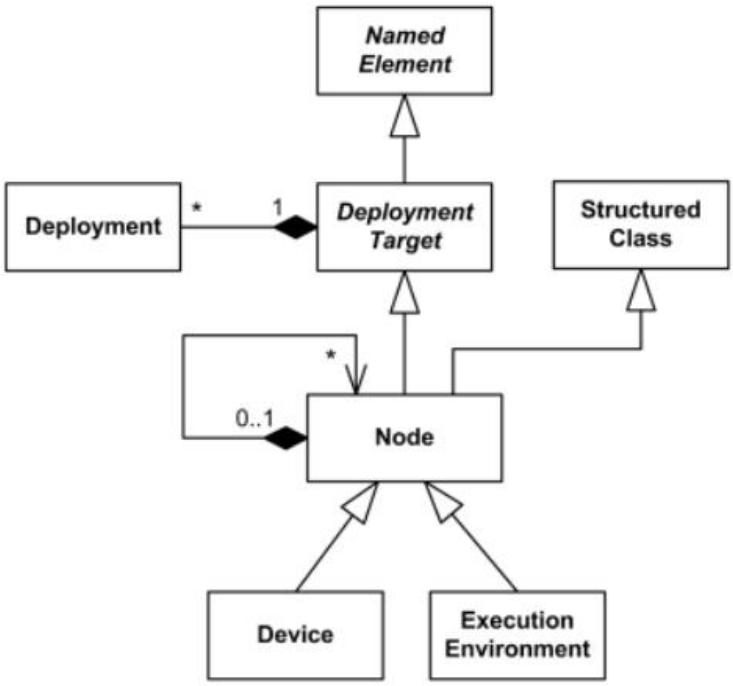


Рис. 4.4 – деплоймент діаграма системи

ВИСНОВКИ ДО РОЗДІЛУ 4

В даному розділі були розглянуті приклади роботи програми з продемонстрованими результатами. Також було проведене юніт тестування системи з наведеними успішними результатами. Був використаний кластер Akka всередині контейнерів Docker для процесу деплойменту системи.

ВИСНОВКИ

В даному дипломному проекті розроблюється система для моніторингу інтернет-медіа ресурсів з елеменатами асинхроного моніторингу та парсингу сайів. В процесі розробки розглядалася актуальність проблеми. Були досліджені існуючи системи та їх недоліки та розглянуті шляхи їх вирішення. Був наданий опис предметної області, розглянуті основні методи розробки даних систем. Визначені основні вимоги до функціоналу та розглянуті основні прецеденти системи, життєві цикли запита користувача та основні реалізації поставлених задач. Були описані основні технології які використовувалися для розробки та їхня реалізація в системі. Була розроблена діаграма класів, їхня функціональна та структурна взаємодія.

Розглянута модель акторів як спосіб реалізації систем моніторингу та використання фреймворку Akka для досягнення поставлених функціональних задач. Також розглянуті основні підходи та методи парсингу сайтів та їх використання у розробці системи. Дана система може використовуватися для вирішення широко спектру різних бізнес задач або для звичайного щоденого пошуку інформації, який є і буде актуальним для усіх нас.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Akka in Action - Raymond Roestenburg, Rob Bakker, and Rob Williams 448 c.
2. Mastering Akka, by Christian Baxter, PACKT Publishing, October 2016
3. Learning Akka, by Jason Goodwin, PACKT Publishing, December 2015
4. Reactive Messaging Patterns with the Actor Model, by Vaughn Vernon, Addison-Wesley Professional, August 2015
5. Developing an Akka Edge, by Thomas Lockney and Raymond Tay, Bleeding Edge Press, ISBN: 9781939902054, April 2014
6. Effective Akka, by Jamie Allen, O'Reilly Media, ISBN: 1449360076, August 2013
7. Регулярні вирази [Електронний ресурс] / в – Режим доступу до ресурсу: https://docs.scala-lang.org/ru/tour/regular-expression-patterns.html.
8. Programming Scala: Scalability = Functional Programming + Objects (O'Reilly), by Dean Wampler and Alex Payne
9. Scala Cookbook: Recipes for Object-Oriented and Functional Programming (O'Reilly) by Alvin Alexander
10. Functional Programming in Scala (Manning), by Paul Chiusano and Rúnar Bjarnason
11. Scala for the Impatient (Addison-Wesley), by Cay S. Horstmann
12. Scala Puzzlers: The fun path to deeper understanding, by Andrew Phillips and Nermin Šerifović (Artima)
13. Scala for Machine Learning: Leverage Scala and Machine Learning to study and construct systems that can learn from data, Second Edition, by Patrick R. Nicolas (Packt)
14. ODIN. Open-Domain-INformer [Електронний ресурс] / ODIN. – 2016. – Режим доступу до ресурсу: https://github.com/clulab/processors/wiki/ODIN-(Open-Domain-INformer).

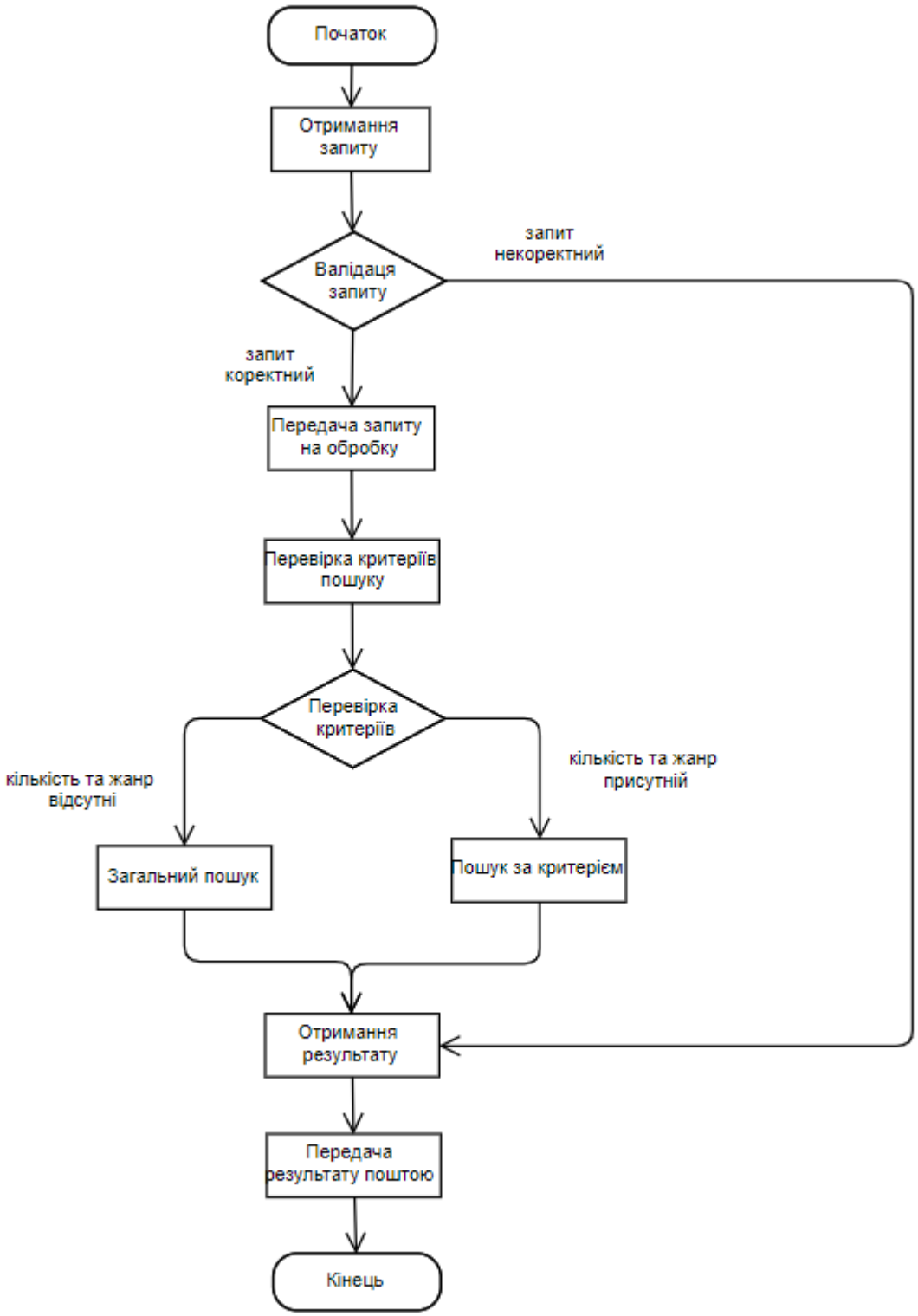
15. Akka [Електронний ресурс] – Режим доступу до ресурсу: https://akka.io/docs/.
16. Моніторинг [Електронний ресурс] – Режим доступу до ресурсу: http://nbuviap.gov.ua/index.php?option=com_content&view=article&id=3498:suchasni-osoblivosti-rozvitku-metodiv-kontent-monitoringu-i-kontent-analizu-informatsijnikh-potokiv&catid=81&Itemid=41

Додаток А

Принципова схема веб-сервісу для моніторингу Інтернет-медіа ресурсів

до дипломного проєкту
на тему: «Веб-сервіс для моніторингу Інтернет-медіа ресурсів»

Київ – 2020 року



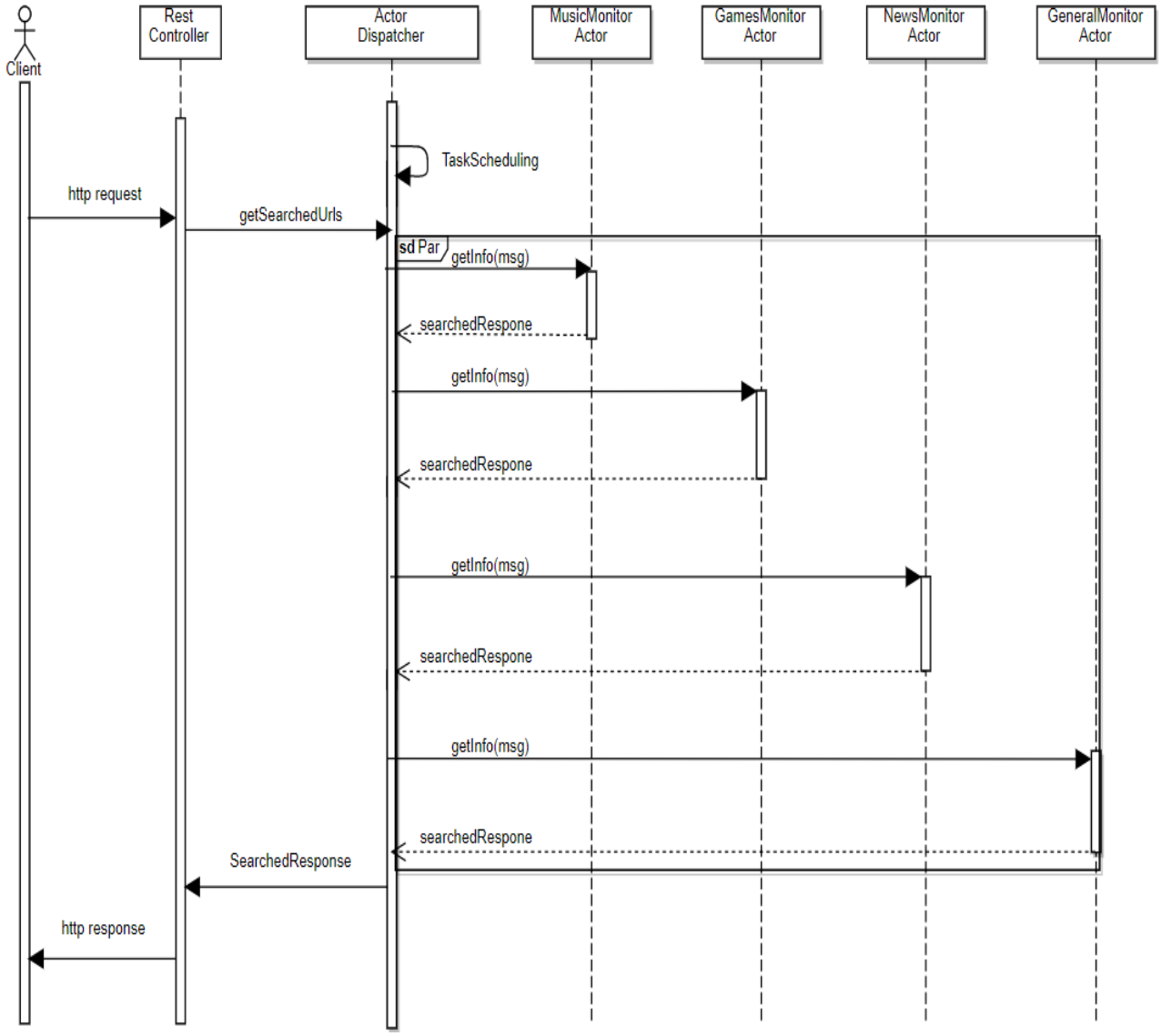
					ІАЛЦ.467100.004 Д1		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*	*Веб-сервіс для моніторингу Інтернет-медіа ресурсів*	*Літ.*	*Маса.*	*Масш.*
Розроб.		*Траэр А. М.*			*Принципова схема алгоритму веб-сервісу для моніторингу Інтернет-медіа ресурсів*			
Перевір.		*Болдак А.О.*						
Т. Конто.						*Аркуш* 1	*Аркушів* 1	
Н. Контр.		*Сімоненко В.П.*				*НТУУ «КПІ», ФІОТ ІО-64*		
Затверд.								

Додаток Б

Структурна схема веб-сервісу для моніторингу Інтернет-медіа ресурсів

до дипломного проєкту
на тему: «Веб-сервіс для моніторингу Інтернет-медіа ресурсів»

Київ – 2020 року



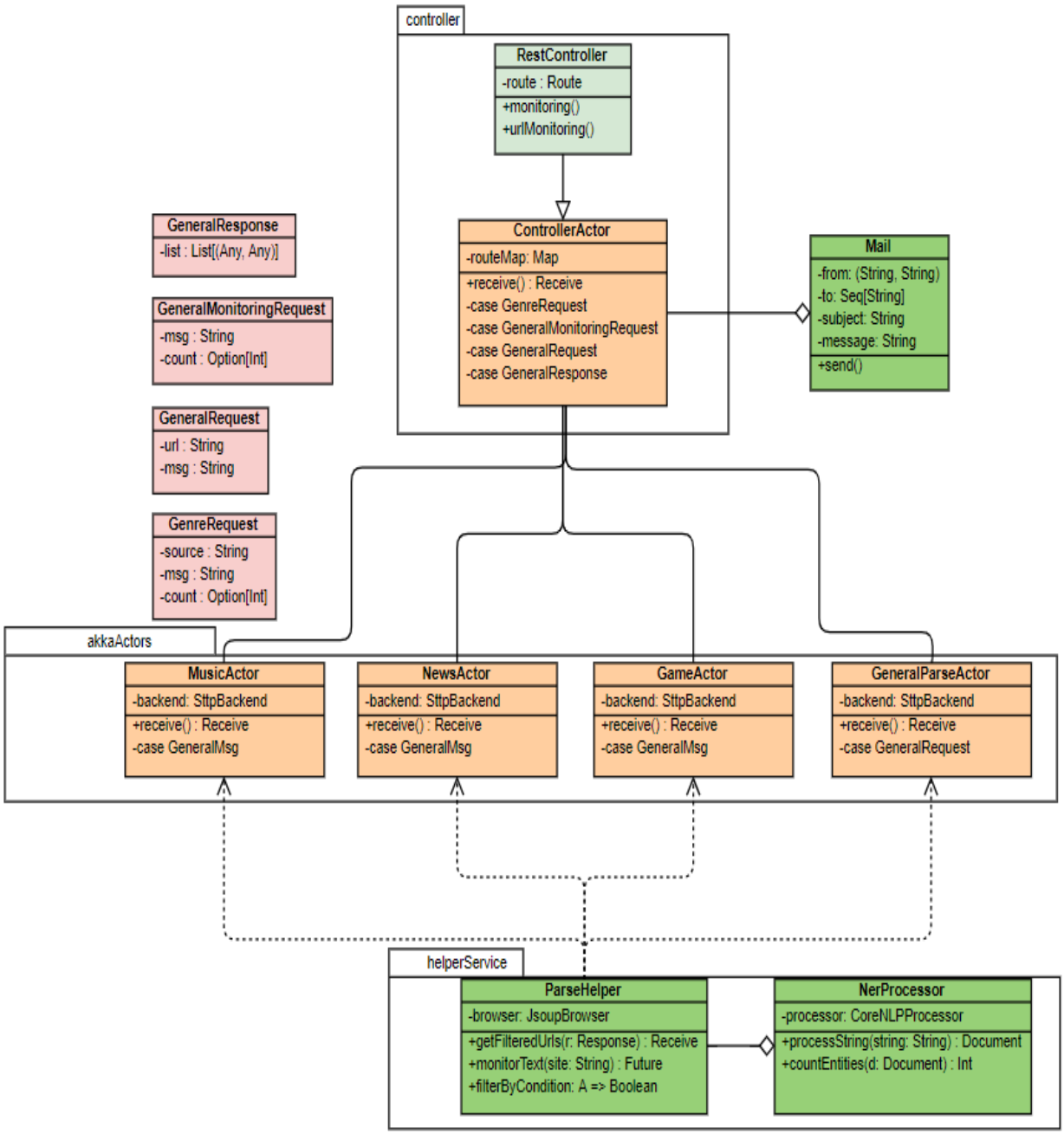
					ІАЛЦ.467100.005 Д2		
					Веб-сервіс для моніторингу Інтернет-медіа ресурсів Структурна схема веб-сервісу для моніторингу Інтернет-медіа ресурсів	Літ.	Маса.	Масш.
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Траэр А. М.				Аркуш 1	Аркушів 1	
Перевір.		Болдак А.О.						
Т. Конто.								
Н. Контр.		Сімоненко В.П.				НТУУ «КПІ», ФІОТ ІО-64		
Затверд.								

Додаток В

Функціональна схема класів веб-сервісу для моніторингу Інтернет-медіа ресурсів

до дипломного проєкту
на тему: «Веб-сервіс для моніторингу Інтернет-медіа ресурсів»

Київ – 2020 року



					ІАЛЦ.467100.006 ДЗ		
					Веб-сервіс для моніторингу Інтернет-медіа ресурсів Функціональна схема класів веб-сервісу для моніторингу Інтернет-медіа ресурсів	Літ.	Маса.	Масш.
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Траэр А. М.				Аркуш 1	Аркушів 1	
Перевір.		Болдак А.О.						
Т. Конто.								
Н. Контр.		Сімоненко В.П.				НТУУ «КПІ», ФІОТ ІО-64		
Затверд.								

Додаток Г
до дипломного проєкту

на тему: «Веб-сервіс для моніторингу Інтернет-медіа ресурсів»

Київ – 2020

ЛІСТИНГ ПРОГРАМИ (СЕРВЕРНА ЧАСТИНА)

```
object RestController {

  object GeneralRequestJsonSupport extends DefaultJsonProtocol with SprayJsonSupport {
    implicit val PortofolioFormats: RootJsonFormat[GeneralRequest] =
jsonFormat2(GeneralRequest)
  }

  def main(args: Array[String]) {

    import GeneralRequestJsonSupport._

    implicit val system = ActorSystem("monitoringSystem")
    implicit val materializer = ActorMaterializer()
    // needed for the future flatMap/onComplete in the end
    implicit val executionContext = system.dispatcher
    val mainActor = system.actorOf(ControllerActor.probs(system), "mainActor")

    val route =
      path("monitoring") {
        get {
          parameters('msg.as[String], 'genre.as[String].?, 'count.as[Int].?) { (msg,
genre, count) =>
           mainActor ! genre.map(value => GenreRequest(value, msg,
count)).getOrElse(GeneralMonitoringRequest(msg, count))
            complete((StatusCodes.Accepted, "monitoring started"))
          }
        }
      }
    val route2 =
      path("urlMonitoring") {
        post {
          entity(as[GeneralRequest]) { request =>
            mainActor ! request
            complete((StatusCodes.Accepted, "monitoring started"))
          }
        }
      }
    val result = route ~ route2

    val bindingFuture = Http().bindAndHandle(result, "localhost", 8080)
```

```
    println(s"Server online at http://localhost:8080/\nPress RETURN to stop...")
    StdIn.readLine() // let it run until user presses return
    bindingFuture
      .flatMap(_.unbind()) // trigger unbinding from the port
      .onComplete(_ => system.terminate()) // and shutdown when done
  }

class ControllerActor(system: ActorSystem) extends Actor {

  import ControllerActor._
  import system.dispatcher

  val gameActor = system.actorOf(GameActor.probs)
  val musicActor = system.actorOf(MusicActor.probs)
  val newsActor = system.actorOf(NewsActor.probs)
  val generalParseActor = system.actorOf(Props[GeneralParseActor])


  val routeMap = Map("Games" -> gameActor, "Music" -> musicActor, "News" -> newsActor)
  var timerCancellable: Option[Cancellable] = None

  override def receive: Receive = {

    case GenreRequest(source, msg, count) =>
      cancelExistingScheduling
      timerCancellable = Some(
        system.scheduler.schedule(120 milliseconds, 3000 seconds ,routeMap(source),
GeneralMsg(msg, count))
      )

    case GeneralMonitoringRequest(msg, count) =>
      cancelExistingScheduling
      routeMap.values.foreach(actor => {
        system.scheduler.schedule(120 milliseconds, 3000 seconds ,actor,
GeneralMsg(msg, count))
      })

    case request: GeneralRequest =>
      cancelExistingScheduling
      timerCancellable = Some(
        system.scheduler.schedule(120 milliseconds, 3000 seconds, generalParseActor,
request)
```

```
      )

    //TODO Aggregator pattern or ask-pipe should be used
    case GeneralResponse(list) =>
      send a Mail(
        from = ("artem.traer@gmail.com", "John Smith"),
        to = Seq("artem.traer@gmail.com"),
        subject = "Monitoring",
        message = "Your urls for monitoring " + "\n" + list.map(_._2).mkString("\n")
      )
  }
  private def cancelExistingScheduling = {
    timerCancellable.foreach(_.cancel())
  }
object ParseHelper {

  val browser = JsoupBrowser()
  import scala.concurrent.ExecutionContext.Implicits.global


  def getFilteredUrls(response: Response[Either[String, String]], count: Option[Int],
find: String) = {
    val stringResponse: String = response.body match {
      case Right(value) => value
      case Left(e) => e
    }

    val jsonObject = JSON.parseFull(stringResponse)
    val further = jsonObject match {
      case Some(map: Map[Any, Map[Any, Any]]) => map("data")("children")
    }
    val mapSet = further.asInstanceOf[List[Map[Any, Map[Any,
Any]]]].map(_.values.last)
    val titlesWithUrl = mapSet.map(elem => (elem.getOrElse("title", "No Title"),
elem.getOrElse("url", "none")))
    /*
          println("FIRST = " + titlesWithUrl.head._1.toString)
          ProcessorExample.processString(titlesWithUrl.head._1.toString)*/

    def filterByCondition : ((Any, Any)) => Boolean = {
      elem => count match {
        case Some(number) => StringUtils.countMatches(elem._1.asInstanceOf[String],
find) >= number
```

```
        case None => StringUtils.containsIgnoreCase(elem._1.asInstanceOf[String],
find)
      }
    }
    titlesWithUrl.filter(filterByCondition).sortWith((a: (Any, Any), b: (Any, Any)) =>
      processString(a._1.asInstanceOf[String]) >
processString(b._1.asInstanceOf[String]) )
  }
  def monitorText(site: String, msg: String, count: Option[Int]):
Future[Option[List[(Any, Any)]]] = {
    Future {
      println("lol")
     count match {
       case Some(number) =>
         if (StringUtils.countMatches(browser.get(site).root.text, msg) >= number)
           Some(List(("Site to visit ", site)))
         else None
       case None =>
         if (StringUtils.containsIgnoreCase(browser.get(site).root.text, msg))
           Some(List(("Site to visit ", site)))
         else None
     }
    }

class GameActor extends Actor {
  import scala.concurrent.ExecutionContext.Implicits.global
  implicit val backend: SttpBackend[Identity, Nothing, NothingT] =
HttpURLConnectionBackend()
  override def receive = {
    case GeneralMsg(find, count)  =>
        val response2 = basicRequest.header("User-Agent", "myapp")
          .get(uri"https://www.reddit.com/r/Games/.json?count=2").send()
      val tuples = getFilteredUrls(response2, count, find)
      val list = List("https://www.pcgamer.com", "https://www.epicgames.com",
"https://www.gamesradar.com/")
      val mySender = sender()

package akkaActors

import akka.actor.{Actor, Props}
import controller.ControllerActor.GeneralResponse
import sttp.client.{basicRequest, _}
```

```
import scala.concurrent.Future
import scala.util.{Failure, Success}


object MusicActor {
  def probs = Props(new MusicActor())
}

class MusicActor extends Actor {

  import scala.concurrent.ExecutionContext.Implicits.global
  implicit val backend: SttpBackend[Identity, Nothing, NothingT] =
HttpURLConnectionBackend()
  import GameActor._
  import helperService.ParseHelper._


  override def receive = {

    case GeneralMsg(find, count)  =>
        val response2 = basicRequest.header("User-Agent", "myapp")
          .get(uri"https://www.reddit.com/r/Music/.json?count=2").send()

      val tuples = getFilteredUrls(response2, count, find)

      val list = List("https://www.music-news.com/", "https://www.nme.com/news/music",
"https://www.billboard.com/")
      val mySender = sender()

      Future.sequence(list.map(site => monitorText(site, find, count))).map(s =>
s.flatten.flatten) onComplete {
        case Success(resultList) =>  mySender ! GeneralResponse(tuples ++ resultList)
        case Failure(exception) => println(exception)
      }
  }

    // Can't add these via fluent API because it produces exceptions
    mail.to foreach commonsMail.addTo
    mail.cc foreach commonsMail.addCc
    mail.bcc foreach (commonsMail.addBcc(_))

    import org.apache.commons.mail.DefaultAuthenticator
```

```
package akkaActors

import akka.actor.Actor
import controller.ControllerActor.{GeneralRequest, GeneralResponse}
import net.ruippeixotog.scalascraper.browser.JsoupBrowser
import org.apache.commons.lang3.StringUtils
import sttp.client.{basicRequest, _}

import scala.util.parsing.json.JSON


class GeneralParseActor extends Actor {

  val browser = JsoupBrowser()

  override def receive = {

    case GeneralRequest(site, msg) =>
      val doc2 = browser.get(site)
      val sWord = StringUtils.containsIgnoreCase(doc2.root.text, msg)
      sender() ! GeneralResponse(List(("result", if (sWord) site else "None")))
  }
}

object send {
  def a(mail: Mail) {
    import org.apache.commons.mail._

    val format =
      if (mail.attachment.isDefined) MultiPart
      else if (mail.richMessage.isDefined) Rich
      else Plain

    val commonsMail: Email = format match {
      case Plain => new SimpleEmail().setMsg(mail.message)
      case Rich => new
HtmlEmail().setHtmlMsg(mail.richMessage.get).setTextMsg(mail.message)
      case MultiPart => {
        val attachment = new EmailAttachment()
        attachment.setPath(mail.attachment.get.getAbsolutePath)
        attachment.setDisposition(EmailAttachment.ATTACHMENT)
        attachment.setName(mail.attachment.get.getName)
        new MultiPartEmail().attach(attachment).setMsg(mail.message)
      }
    }
```